Post Training in Deep Learning with Last Kernel

Thomas Moreau
Centre de Mathematiques et Leurs Applications
ENS Paris-Saclay, CNRS, Université Paris-Saclay,
94235, Cachan, France.

Julien Audiffren
Centre de Mathematiques et Leurs Applications
ENS Paris-Saclay, CNRS, Université Paris-Saclay,
94235, Cachan, France.

Abstract—One of the main challenges of deep learning methods is the choice of an appropriate training strategy. In particular, additional steps, such as unsupervised pre-training, have been shown to greatly improve the performances of deep structures. In this paper, we introduce a new training step, the post-training, which takes place after the training and where only specific layers are trained. In particular, we focus on the particular case – named Last Kernel – where only the last layer is trained. This step aims to find the optimal use of data representation learned during the other phases of the training. We show that Last Kernel can be effortlessly added to most learning strategies, is computationally inexpensive, does not cause overfitting and often produces significant improvement. Additionally, we show that with commonly used losses and activation functions, Last Kernel solves a convex closed optimization problems, offering rapid convergence – or even closed-form solutions.

I. INTRODUCTION

In the recent years, Deep Learning algorithms have been shown to be successful at solving arduous Artificial Intelligence related tasks, such as Computer Vision (see e.g. [1]) or Natural Language Processing (see e.g. [2]). Part of this success can be explained by the recent progresses in training strategies, allowing the use of deeper structures which have shown to repeatedly outperform comparable shallow models (see e.g. [3], [4]).

Indeed, one of the main challenges of the deep learning methods is how to efficiently solve a highly complex and non convex optimization problem. Consequently, the choice of an appropriate training strategy is of paramount importance, as small mistakes can drive the algorithm into a highly suboptimal local minimum, resulting into poor performances (see e.g. [5]).

A frequently used approach is the pre-training of the network, based on the iterative construction of the layers, using unsupervised criteria such as the stacking of auto associators (see e.g. [6], [3]). The network is then fine-tuned using gradient-based optimization to optimize all the layers simultaneously. Deep networks trained with these strategies have been applied successfully to many tasks, such as classification tasks [5], [7], regression [8], robotics [9] or information retrievals [10].

In addition to the progress of the training strategies, those works have also shed some light onto the role of the different layers (see e.g. [11], [12]). For instance, it has been shown that the first layers of a deep neural networks tend to learn general characteristics of the dataset which can be reused in other architectures, independently of the solved task. Those layers are qualified as general. For example, in image processing, those first layers of convolutional network exhibit features similar to Gabor filters and color blob (see e.g. [1]). Conversely, the last layers of the network greatly depend on the chosen dataset and task, and are referred to as specific layers [13]. It is interesting to note that while pre-training of a deep learning network focuses on the general layers, the training itself generally involves all the layers, i.e. the general and the specific ones.

While Deep Learning achieves better results than shallow structures, the later are generally easier to train and more stable — i.e. less dependent to initialization. For instance, kernel methods – by separating data representation and learning – produce powerful and easy to use algorithms, such as kernelized support vector machines [14].

The goal of this paper is to use these ideas of separating representation learning and statistical analysis to improve training strategies of deep structures by introducing a new step, the post-training, where only the specific layers are trained. Since the general layers are frozen, this step will use the data representation learned during both pre-training and training to solve the desired task, by allowing the specific layers to learn the solution to the corresponding statistical model.

In particular, we chose to study the case where only the last layer is not frozen, which we call Last Kernel. This choice is motivated by the following ideas, which are developed in this paper. First, while the exact limit between general and specific layers may be difficult to find [13], the last layer is always the most specific. Second, in this setting the post training problem can be seen as a learning problem method with a given feature function. Moreover, in this setting, learning the weights of the last layer corresponds to learning the weight of the kernel associated to the given feature function. The post training scheme can thus be interpreted in light of the different results for kernel models.

In this paper we make the following contributions:

• We introduce Last Kernel, a post training step, where all layers except the last are frozen. This method can be
applied after any traditional optimization scheme for deep networks.

- We show that Last Kernel is easy to use, and can be effortlessly added to most learning strategies.
- We highlight the link existing between this method and the kernel techniques.
- We experimentally show that Last Kernel is computationally inexpensive, and does not overfit and often produces significant improvement.
- We prove that with commonly used losses and activation function, Last Kernel solve a convex closed optimization problems, offering rapid convergence – or even close form solutions.

The paper is organized as follows: Section II discusses existing papers that relate to our work, including transfer learning, pre-training and kernel approaches to deep learning. Section III introduces the post-training phase and Last Kernel, and discusses its relation with kernel methods. Section IV presents our numerical experiments and Section V discusses those results.

II. RELATED WORK

The study of the interrelation between kernels and deep learning is hardly new. For instance, [12] used kernels to study empirically the layer-wise evolution of the representation of the data in deep learning networks; [15] introduced new types of kernels that were derived from the computations of neural networks and that can be used in different architectures. The authors in [16] used ideas from the structure of neural network to propose a multilayer multiple kernels learning version.

Previous works have studied the features learned by the different layers of a deep learning network, and particularly their influence on the performances of the network. For example, [13] highlighted the different roles of the layers with respect to the final features, and the transferability of said features, by using the concept of freezing layers, i.e. layers whose weights are fixed, in transfer learning.

Research on learning strategies for deep structure has produced an abundant literature. [5] and [17] developed the idea of training neural network architectures layer by layer in the unsupervised pre-training initialization; [18] studied the influence of the pre-training and proposed some variants to this method.

One of the papers closest to our work is arguably [19], where the authors used the relation between kernels and the last layer of a deep neural network to propose a novel regularization function for visual recognition.

However, and to the best of our knowledge, the approach of post-training, i.e. the training of the specific layers in the final step of the supervised learning that we present in this paper, is new.

III. POST TRAINING WITH LAST KERNEL

Notation. In the following, $\mathcal{X}_1, \ldots, \mathcal{X}_L$ are Polish spaces – typically $\mathbb{R}^d$ with $d > 0$ – that correspond to the input spaces of the different layers, $\mathcal{Y} = \mathcal{X}_{L+1}$ is the output space (with either $\mathcal{Y}$ is finite or $\mathcal{Y}$ is a Hilbert space) and $\mathcal{D} = (x_i, y_i)_{i=1}^N$ are the elements of the dataset, drawn from the random variable $(x, y)$ in $(\mathcal{X}_1, \mathcal{Y})$. Also, let $\phi_j : \mathcal{X}_j \mapsto \mathcal{X}_{j+1}$, for $1 \leq l \leq L$, be the applications which respectively computes the output of the $l$-th layer of the network, using the output of the $l-1$-th layer. Finally, let $\Phi_L = \phi_L \circ \cdots \circ \phi_1$ be the mapping of the full network from $\mathcal{X}_1$ to $\mathcal{Y}$, with $L \in \mathbb{N}$ denoting the number of layers of the network.

Last Kernel is a special case of post training where the last layer is trained using the features learned by the previous layers during the training phase. Figure 1 illustrates this process. The objective is to find an optimal use of the learned features.

A. Formalizing Last Kernel

Let $\ell : \mathcal{Y} \times \mathcal{Y} \mapsto \mathbb{R}^+$ denotes a convex and continuous loss function. The objective of the neural network training is to find weights parametrizing $\Phi_L$ that solves the following problem:

$$\arg \min_{\Phi_L} \mathbb{E}_{(x,y)} [\ell(\Phi_L(x), y)] \, .$$

(1)

Let assume that the last layer of the network is fully connected and let denote by $f$ the activation function and by $W_L$ the weights. Then, since we have

$$\Phi_L(\cdot) = f(\Phi_{L-1}(\cdot) W_L^T) \, ,$$

[1] can then be rewritten as

$$\arg \min_{\Phi_{L-1}, W_L} \mathbb{E}_{x,y} [\ell (f(\Phi_{L-1}(x) W_L^T), y)] \, .$$

(2)
Algorithm 1 Last Kernel

Require: Criteria $C_1, C_2$ and regularization $\lambda > 0$
1: Training:
    Train the global network $\Phi_L$ with any classical optimization technique until $C_1$ is met.
2: Post-training:
    Solve the optimization problem
    \[
    \arg\min_{W_L} \frac{1}{N} \sum_{i=1}^{N} \tilde{\ell}(\Phi_{L-1}(x_i)W_L^T, y_i) + \lambda||W_L||^2_2
    \]
    for $W_L$ with the previous layers $\Phi_{L-1}$ frozen, until $C_2$ is met.

It is important to note that for several popular choices of activation function $f$ and loss $\ell$, such as softmax and cross entropy respectively, this problem with $\Phi_{L-1}$ is convex respectively to $W_L$ (see e.g. Proposition A.1). In this case, classical convex optimization techniques can efficiently produce accurate approximation of the optimal weights. Last Kernel leverages this observation to obtain a performance boost in the post-training. It proceeds as follows:

1) First, a regular training strategy is applied to the network on the empirical loss
    \[
    \arg\min_{\Phi_L} \frac{1}{N} \sum_{i=1}^{N} \ell(\Phi_L(x_i), y_i). \tag{3}
    \]
    This regular training might last for a fixed number of epochs, or can stop after a criterion is verified (such as early stopping [20]). This step aims to obtain interesting features to solve the initial problem, as in any usual deep learning training. It is important to note that any type of training strategy can be used here, including gradient bias reduction techniques, such as Adagrad [2], or regularization strategies, e.g. Dropout (see [21]).

2) Then, for the post-training, the $L-1$ first layers are frozen, and only the last layer of the network, $\phi_L$, is trained by minimizing
    \[
    \arg\min_{W_L} \frac{1}{N} \sum_{i=1}^{N} \tilde{\ell}(\Phi_{L-1}(x_i)W_L^T, y_i) + \lambda||W_L||^2_2, \tag{4}
    \]
where $\tilde{\ell}(x, y) := \ell(f(x), y)$. The idea here is to add a learning step once the representation is fixed: the feature function $\Phi_{L-1}$ embeds the inputs into $\mathcal{X}_L$ and the post training permits to learn weights that to solve the task of interest based on this representation. It is important to note that Dropout should not be applied on the previous layers of the network during the post-training, as it would lead to changes in the feature function $\Phi_{L-1}$. This step is computationally faster than the initial training as there is no need for back propagation to update only $W_L$.

Convexity. One of the advantages of Last Kernel is that for reasonable choices of activation and loss functions, (4) is convex and thus can be minimized efficiently. As such, theoretical guarantees such as converge rates in $\mathcal{O}(\sqrt{1/N})$ apply for gradient descent technique and reasonable choices of learning rate and regularization (see e.g. [22]). Those results ensure that Last Kernel will produce an improvement of the performances of the network.

Adding a regularization. The addition of a strongly convex regularization term to the objective function (4) during the post-training can improve the speed of convergence of Last Kernel as well as its accuracy. In particular, the regularization
    \[
    \Psi(W) = \lambda||W||^2_2,
    \]
seems to produce good empirical results (see Section IV), and its choice is motivated by the relation with kernels described in Subsection III-B. Algorithm 1 presents Last Kernel using this regularization.

The choice of the criteria $C_1, C_2$ of Last Kernel, as well as the influence of the parameter $\lambda$, are discussed in Section IV and Section V.

B. Link with kernels

Consider the case where $\mathcal{X}_L = \mathbb{R}^N$ and $\mathcal{X}_{L+1} = \mathbb{R}$. In particular, if we define a kernel $k$ as
    \[
    k : \mathcal{X}_L \times \mathcal{X}_L \mapsto \mathbb{R} \quad (x_1, x_2) \mapsto \langle \Phi_{L-1}(x_1), \Phi_{L-1}(x_2) \rangle
    \]
then $k$ is the kernel associated with the feature function $\Phi_{L-1}$. It is easy to see that this kernel is continuous positive definite and that the function
    \[
    g_W : \mathcal{X}_L \mapsto \mathcal{X}_{L+1} \quad x \mapsto \langle \Phi_{L-1}(x), W \rangle
    \]
belongs by construction to the Reproducing Kernel Hilbert Space (RKHS) $\mathcal{H}_k$ generated by $k$. It is interesting to remark that
    \[
    \arg\min_{g \in \mathcal{H}_k} \frac{1}{N} \sum_{i=1}^{N} \tilde{\ell}(g(x_i), y_i) + \lambda||g||^2_{\mathcal{H}_k},
    \]
is a classical equation in kernel theory. With mild hypothesis on $\tilde{\ell}$, the generalized representer theorem (23) can be applied. As a consequence, $\exists!\alpha^* \in \mathbb{R}^N$ such that
    \[
    g^* := \arg\min_{g \in \mathcal{H}_k} \frac{1}{N} \sum_{i=1}^{N} \tilde{\ell}(g(x_i), y_i) + \lambda||g||^2_{\mathcal{H}_k} = \sum_{i=1}^{N} \alpha^*_i k(X_i, \cdot) = \sum_{i=1}^{N} (\alpha^*_i \Phi_{L-1}(x_i), \Phi_{L-1}(\cdot)). \tag{5}
    \]
By rewriting (5), we have that $g_{W^*}$, with
    \[
    W^* = \sum_{i=1}^{N} \alpha^*_i \Phi_{L-1}(x_i), \tag{6}
    \]
is the optimal solution to the original optimization problem.
The problems (5) and (4) only differ in the choice of the regularization norm. By definition of the RKHS norm, we have that
\[ \|g_W\|_H = \inf \{ \|v\|_2, \langle v, \Phi_{L-1}(x) \rangle = g_W(x) \text{ } \forall x \in \mathcal{X}_1 \} . \]

Consequently, we have that \( \|g_W\|_H \leq \|W\|_2 \), with equality when \( \text{Vect}(\Phi_{L-1}(\mathcal{X}_1)) = \mathcal{X}_L \). In this case, the norm induced by the RKHS is equal to the \( L_2 \)-norm. As the input space is usually in a far higher dimensional space than the output space, and since the neural network structure generally enforces the independence of the features, the aforementioned criterion is often met. Consequently, the \( L_2 \)-norm is a good proxy for the RKHS-norm.

**Close-form solution.** In the particular case where \( \ell(y_1, y_2) = \|y_1 - y_2\|^2 \) and \( f(x) = x \), (5) can be reduced to a classical Kernel Ridge Regression problem. In this setting, \( W^* \) can be computed by combining (6) and
\[
\alpha^* = (\Phi_{L-1}(D)\Phi_{L-1}(D)^T + \lambda I)^{-1} y ,
\]
where \( \Phi_{L-1}(D) \) represents the matrix of the input data \( \{x_1, \ldots, x_N\} \) embedded in \( \mathcal{X}_L \), \( Y \) is the matrix of the output data \( \{y_1, \ldots, y_N\} \) and \( I \) is the identity matrix in \( \mathbb{R}^N \). This result is experimentally illustrated in Subsection IV-A. Although datasets are generally too large for (7) to be computed, it is worth noting that some kernel methods, such as Random Features ([24]), can be applied to compute approximations of the optimal weights during the post-training.

We emphasis that \( W^* \) is related to the optimal solution for the problem (5) and should not be confused with \( W^*_\ell \), the optimum of (4). However, the two problems differ only in their regularization, which are closely related (see the previous paragraph). Thus \( W^* \) can thus be seen as an approximation of the optimal value \( W^*_\ell \) and is empirically evaluated in Subsection IV-A.

**Multi-dimensional output.** Most of the previously discussed results related to kernel theory still holds for multi-dimensional output spaces, i.e. \( \dim(\mathcal{X}_{L+1}) = d > 1 \), using multitask or operator valued kernels (see e.g. [25]). Hence the previous remarks regarding Last Kernel can be easily extended to multi-dimensional outputs, encouraging the use of post training in most settings.

### IV. Experimental results

This section provides numerical arguments to analyze Last Kernel and its performances. All the experiments were run using Python and Tensorflow, and the training was performed using GradientDescent. The code to reproduce the figures is available online\(^{1}\).

---

\(^{1}\)The code can be found at [https://github.com/tomMoral/LastKernel](https://github.com/tomMoral/LastKernel)
Fig. 2: Evolution of the performance of the neural network on regression tasks. In each figure, we plotted the results of the usual training (blue), the results of Last Kernel started at different time (purple) and the results using the weights $W_L = W^*$ computed using equation (7) (red).

Fig. 3: Illustration of the neural network structure used for CIFAR-10. The last layer, represented in blue, is the one trained using Last Kernel.

B. CIFAR-10

To assert the performances of our post training method, we tested our method on the Cifar10 dataset [27]. This dataset is composed of 60,000 images 32x32, representing one object in 10 classes. We used the default architecture proposed for Cifar10 by Tensorflow in our experiments. This architecture is based on the original architecture proposed in [27], composed of 5 layers. The network is composed of 2 convolutional layers with relu non linearities, max pooling and local response normalization, followed by 2 fully connected layers with relu activation (see Figure 3). The last layer is a fully connected linear layer with softmax activation and we used the cross entropy cost for training.

The network was trained for $100k$ iterations, with batches of size 128, using the gradient descent and an exponential weight decay for the learning rate. Then, we ran Last Kernel post training for $20k$ iterations, starting from the weights of the network saved every $10k$ iterations. The regularization parameter $\lambda$ was set to 0.001, but its value did not have significant impact on the observed results. Figure 4 presents the evolution of the training error relatively to the number of iterations.

This experiment illustrates the effect of the post training for different criteria $C_1$, i.e. the starting time of the post-training. The post training improved the test performance after less than 1000 iterations, which is significantly faster than the continuation of the classical optimization. It is also important to note that this improvement was significant when the training was stopped early, and did not degrade the performances of the network even when used at very large times. The training cost did not drop as significantly as the test error with Last Kernel, reducing the risk of overfitting the training set.

Those results were reproduced with different architectures. We performed the same experiment with other neural networks structures, including networks with additional fully connected layers or with additional convolutional layers. Those results indicate that our observations are not tied to certain type of architecture. Additionally, the improvement of performances originating from Last Kernel was constant, regardless of the regularization used during the training phase – including dropout. Finally, it is worth noting that different values of $\lambda \in [10^{-6}, 10^3]$ were tested in this experiment. Last Kernel appeared to be robust to the choice of reasonable values of $\lambda$ (i.e. $10^{-6} \leq \lambda \leq 10^{-2}$).

V. DISCUSSION

The experiments presented in Section IV show that Last Kernel improves the performances of the networks most of the time. This gain is even more significant when the network did not fully converge prior to the post-training. Indeed, in this case, the training cost quickly drops to attain a plateau, close to the minimum. This behavior is illustrated in all the experiments. For models where the training is stopped early, the improvement generated by Last Kernel tends to be large. For instance, when the network is trained for less than 80k iterations in our CIFAR-10 experiments, the error rate decreases quickly before reaching a plateau during the post-training.

On the other hand, if the post-training is done after the convergence of the network, the improvement is marginal. However, it is very important to note that regardless of the time Last
Kernel does not produce no over-fitting. This might stem from the fact that the post-training optimization is much simpler than the original problem and lies in a small-dimensional space — which, combined with the added $\mathcal{L}^2$-regularization, efficiently prevents overfitting. Thus, the post training can be applied to any trained network, without prerequisites about how optimized it is since it does not degrade its performances. In particular, this suggests that Last Kernel might be a good addition to the early stopping technique.

Some significant additional benefits of Last Kernel are related to the convexity of the resulting minimization problem for common choices of loss and activation function (such as cross entropy and soft max). When combined with the added $\mathcal{L}^2$-regularization, the problem becomes strongly convex, ensuring rapid convergence and strong generalization property (see e.g. [20]). Additionally, the strong convexity of the problem significantly reduces the variance of both train cost and test error once the minimum is reached.

It is important to note that the computational cost of the post training scheme is very low compared to the one of the full training. The different experiments highlight that only a low number of iteration is necessary to attain a local minimum.

Indeed, in all the experiments, the system rapidly reaches a local minimum and this minimum is stable as following iterations yields negligible performance changes. This might originate from the small number of parameters involved in problem [4] compared to the dimensionality of the original optimization problem. Moreover, when the cost is convex, the optimization is guaranteed to converge rapidly. Also, gradient computations are much simpler for the last layer than for the other layers, as there is no need to chain high dimensional linear operations, contrarily to regular backpropagation. Thus, the computational complexity of Last Kernel iterations is reduced compared to the complexity of full iterations.

Finally, Last Kernel can be used to reduce even more the risk of over-fitting. The solutions obtained with Last Kernel have a lower generalization error as they reach similar performances with worst training cost, as it is illustrated in all three experiments. Thus, reducing the optimization space at the end of the training seems to have a regularization effect on the network performances.

Choice of parameters. As discussed earlier, the choice of the regularization parameter $\lambda$ is important. This choice results from a tradeoff between computational and statistical perfor-
mans. Taking $\lambda$ very large make the optimization problem simpler but reduces the explanatory capacity of the networks whereas if $\lambda$ is too small, the optimization problem become less strongly convex, thus computationally more expensive. Overall, our experiments highlighted that Last Kernel produces significant results for any choice of $\lambda$ reasonably small ($i.e.$ $10^{-6} \leq \lambda \leq 10^{-2}$). It is worth noting that the computational cost of the optimization remains negligible compared to the one of the full optimization even for small values of $\lambda$.

As illustrated in Section IV, Last Kernel produces interesting results, independently of the starting time. Thus the post-training can be applied for any choice of stopping criterion $C_1$, such as early stopping, one pass strategy or fixed time horizon. Moreover, since the post-training converges with little variance, any gradient based criteria can be used for $C_2$ (reasonable fixed horizon criteria can also be used as the convergence is rapid).

Relation with other ideas. The results from Subsection IV-A show an interesting behavior. The post training, started from any state of the network, achieves a test error equivalent to a fully trained network. This highlights the importance of the specialization of the last layer of the network. The post training optimizes the classification performance for a given kernel. When Last Kernel is used on untrained network, the previous layer perform a random embedding of the data – and the simple datasets used in these experiments can be efficiently represented by these embedding. This can be linked to the results from [28] where the authors illustrate that the network architecture sometime suffices to extract interesting features, without training. Thus, the choice of $C_1$ reflects the need of specific features compared to general one obtain from the network architecture. The post training is then just a step to ensure that the best model is build from the selected features.

VI. CONCLUSION

In this work, we showed that the addition of a post training phase, during which only the last layer is trained, is computationally inexpensive and can be beneficial to most neural network structures. We chose to focus on post-training solely the last layer, as it is the most specific layer in the network and the resulting problem is strongly convex under reasonable prerequisites, resulting in a rapid and robust improvement of the performances. The relationship between the number of layers frozen in the post-training and the resulting improvements might be an interesting direction for future works.

Additionally, the link between Last Kernel and kernel methods is interesting as it suggests that using recent kernel techniques could lead to improvement the performance of the deep networks last layer training. Future works might study how methods such as sketching could be use to approximate optimal weights for large kernels with statistical guarantees – allowing to solve [4] more efficiently.
we have
\[
\frac{\partial F(W)}{\partial W_{m,n}} = -\sum_{i=1}^{M} \delta_{ij} \frac{1}{P_i(W)} \frac{\partial P_i}{\partial W_{m,n}}
\]
\[
= x_n \left( \sum_{i=1}^{M} \delta_{ij} P_m(W) - \delta_{mj} \right)
\]
\[
= x_n \left( P_m(W) - \delta_{mj} \right),
\]
hence
\[
\frac{\partial^2 F(W)}{\partial W_{m,n} \partial W_{p,q}} = x_n \left( \frac{\partial P_m}{\partial W_{p,q}} \right),
\]
\[
= x_n x_q P_m(W) \left( \delta_{m,p} - P_p(W) \right).
\]
Hence the following identity
\[
H(F) = P(W) \otimes (XX^T)
\]
where $\otimes$ is the Kronecker product, and the matrix $P(W)$ is defined by $P_{m,p} = P_m(W) \left( \delta_{m,p} - P_p(W) \right)$. Now since $\forall 1 \leq m \leq M$,
\[
\sum_{p=1, p \neq m}^{M} |P_{m,p}| = P_m(W) \sum_{p=1, p \neq m}^{M} P_p(W)
\]
\[
= P_m(W) \left(1 - P_m(W) \right)
\]
\[
= P_{m,m}
\]
The Diagonal dominance theorem implies that $P(W)$ is positive semi definite. Since $XX^T$ is positive semi definite too, their Kronecker product is also positive semi definite, hence the conclusion.

\[ \Box \]

APPENDIX

Proposition A.1 (convexity). \( \forall N, M \in \mathbb{N} , \forall X \in \mathbb{R}^N, \forall j \in [1, M], \) the following function \( F \) is convex:

\[
F : \mathbb{R}^{N \times M} \rightarrow \mathbb{R}
\]
\[
W \rightarrow \log \left( \sum_{i=1}^{M} \exp(XW_i) \right) - \sum_{i=1}^{M} \delta_{ij} \log \left( \exp(XW_i) \right).
\]
where \( \delta \) is the Dirac function, and \( W_i \) denotes the \( i \)-th row of \( W \).

Proof. Let \( P_i(W) = \frac{\exp(XW_i)}{\sum_{j=1}^{M} \exp(XW_j)} \).

then
\[
\frac{\partial P_i}{\partial W_{m,n}} = \begin{cases} 
- x_n P_i(W) P_m(W) & \text{if } i \neq m \\
- x_n P_m(W)^2 + x_n P_m(W) & \text{otherwise}
\end{cases}
\]

Noting that
\[
F(W) = - \sum_{i=1}^{M} \delta_{ij} \log \left( P_i(W) \right),
\]