DIRECT ESTIMATION OF WEIGHTS AND EFFICIENT TRAINING OF DEEP NEURAL NETWORKS WITHOUT SGD

Nima Dehmamy[⋆]*, Neda Rohani*[⋆⋆]*, Aggelos K Katsaggelos*[⋆⋆]

[⋆] CCNR, Department of Physics, Northeastern University. Boston, MA
[⋆⋆] Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL

ABSTRACT

We argue that learning a hierarchy of features in a hierarchical dataset requires lower layers to approach convergence faster than layers above them. We show that, if this assumption holds, we can analytically approximate the outcome of stochastic gradient descent (SGD) for each layer. We find that the weights should converge to a class-based PCA, with some weights in every layer dedicated to principal components of each label class. The class-based PCA allows us to train layers directly, without SGD, often leading to a dramatic decrease in training complexity. We demonstrate the effectiveness of this by using our results to replace one and two convolutional layers in networks trained on MNIST, CIFAR10 and CIFAR100 datasets, showing that our method achieves performance superior or comparable to similar architectures trained using SGD.

1. INTRODUCTION

Deep neural networks (DNN) are useful for tasks which can be broken down into a hierarchy of smaller tasks, sometimes requiring exponentially fewer parameters than their shallow counterparts [1]. Training DNN is generally done using SGD, where backpropagation needs to be done sequentially, layer by layer, and cannot be parallelized. Thus, training deep networks via SGD can be slow and avoiding SGD for some layers could make training DNN more efficient.We show here that, if a hierarchy of features exists in the data, weights for some layers may be approximated without SGD. Learning to classify natural scenes becomes more efficient if they are broken down into low and high-level features [2]. For example faces become easier to classify if the system starts by learning line and curve segments, combining them to make eyes, nose and mouth shapes, and so on. The key point here is that, learning low-level features (e.g., eye shape) should not require knowledge of high-level features (e.g., face composition) at the same time. Therefore, there must exist a way to setup the training dynamics of DNNs such that learning low-level features occurs earlier than high-level features. Since lower layers (closer to input) in a DNN seem to learn low-level features [3], this must mean that lower layers converge more rapidly than higher layers. Fig. 1 shows three examples from MNIST, CIFAR10, and CIFAR100 datasets. The networks trained on these datasets consist of input layer, four dense layers with the same number of hidden units and an ouput classification layer. As can be observed in Fig. 1, in all three networks at early stages, the convergence rate of most lower layers is higher. In terms of the learning dynamics, this means that the dynamics of weights of higher layers play little role in training of lower layers. If this effective "decoupling" of learning dynamics of layers occurs, SGD should be able to train the network in a hierarchical fashion, starting from low-level features. This decoupling greatly simplifies the SGD equations, allowing us to estimate the outcome of the training. Thus, the central question that arises is: What low and high-level features are learned when this decoupling occurs? Examining the SGD equations near convergence we find that this approximate decoupling of layer dynamics would allow us to estimate the outcome of SGD analytically. The final result takes a very simple form, namely that a class-based PCA should be a good approximation of optimal weights, at least for lower layers.

2. SGD NEAR CONVERGENCE

We consider the classification of N inputs $X = (x_1, ..., x_N)$ into C classes by N label vectors $Y = (y_1, ..., y_N)$ using an n layer neural network. The dimension of the output of layer k is $d^{(k)}$. The output of layer k is

$$h^{(k)} = f\left(\tilde{h}^{(k)}\right), \qquad \tilde{h}^{(k)} = {w^{(k)}}^T h^{(k-1)} + b^{(k)} \tag{1}$$

where $f(\cdot)$ is the activation function, $\tilde{h}^{(k)}$ is the "raw output", and $w^{(k)}$ and $b^{(k)}$ are the weights and biases of layer k, respectively. $h_i^{(0)} = x_i$ is the ith input and $h_i^{(n)}$ is the output of the network. The last layer is the classification layer with a softmax activation and categorical cross-entropy cost function $g(h^{(n)}, y)$. The SGD equations are

$$\frac{\delta b^{(k)}}{\delta N} = -\varepsilon \frac{\delta g}{\delta b^{(k)}} = -\varepsilon \left(\frac{\partial g}{\partial \tilde{h}^{(n)}}\right)^T {A^{(k+1)}}^T, \tag{2}$$

$$\frac{\delta w^{(k)}}{\delta N} = -\varepsilon \frac{\delta g}{\delta {w^{(k)}}^T} = -\varepsilon h^{(k-1)} \frac{\delta g}{\delta b^{(k)}}, \tag{3}$$

where N is the number of data points processed, δN is the size of the mini-batch processed in each step, $\delta w \equiv w(N+\delta N) - w(N)$, and ε is the learning rate, which can be dynamically adjusted. The matrix $A^{(k+1)}$ follows from backpropagation (Fig. 2 top left)

$$A^{(k)} \equiv \prod_{m=k}^{n-1} \left(\frac{\partial \tilde{h}^{(m+1)}}{\partial \tilde{h}^{(m)}} \right)^T = \prod_{m=k}^{n} \tilde{w}^{(m)}, \tag{4}$$

$$\tilde{w}^{(m)} \equiv \mathrm{diag}\left(f'\left(h^{(m-1)} \right)^T \right) w^{(m)}. \tag{5}$$

with $\mathrm{diag}(f')$ representing a diagonal matrix with the derivative of the activation function of the hidden layers on the diagonal.

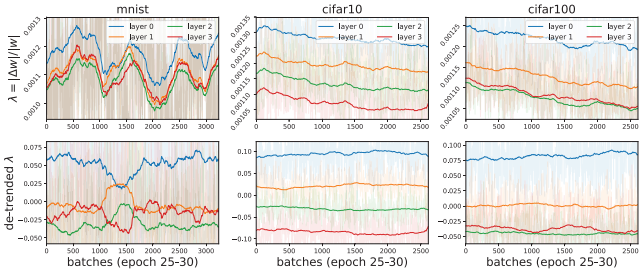

Fig. 1. Convergence rate $\lambda = \|w(N+\delta N) - w(N)\|/\|w(N)\|$ of weights w in different layers on 3 datasets ($\delta N = 64$ is the batch size). The de-trended λ is $\lambda/\overline{\lambda} - 1$, where $\overline{\lambda}$ is the average rate of all layers at a given time step. The solid curves are moving averages of the actual rates (shown with lower opacity). All architectures are an input layer, followed by four fully-connected layers, with the same number of hidden units (81 for MNIST and 100 for CIFAR10 and CIFAR100), and an output layer. The plots show the convergence of the four hidden layers. The convergence rate of lowest layers is generally higher than that of the upper layers in all three datasets. This order in convergence rates changes in later stages of the training.

We assume that the dataset has a clear hierarchy (e.g., faces) and that the architecture of the DNN is designed in a way that it can learn the hierarchy. We are also assuming that lower layers converge faster and learn lower-level features, thus allowing higher layers to learn high-level features by combining learned low-level features. We will now focus on a stage of the training during which layer k is approaching a local minimum, whereas higher layers $m > k$ are still far from convergence. Since layer k is near a local minimum, the gradients $\delta g/\delta b^{(k)}$ and $\delta g/\delta w^{(k)}$ become small. Thus, from (3), $\left\| \partial g/\partial \tilde{h}^{(n)} \right\|$ must be small, fluctuating stochastically around the local minimum. However, note that, while the gradients are small near a local minimum, to ensure lower layers converge faster we expect the rate of convergence, $\lambda = \|\delta w\|/\|w\|$, to be *larger* for lower layers k compared to higher

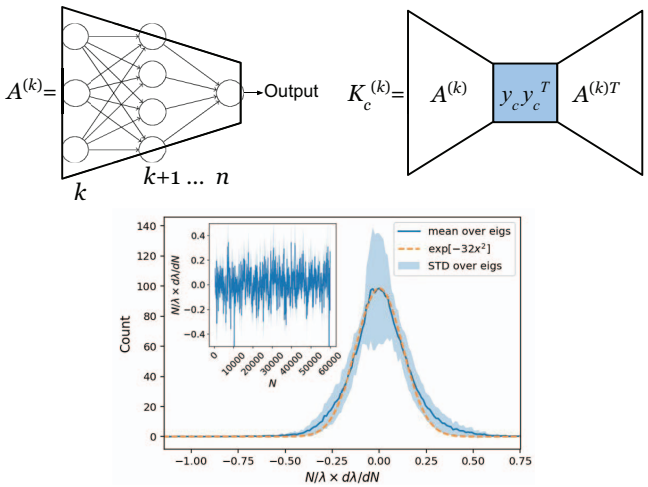

Fig. 2. **Top, Left:** Schematic of the operator $A^{(k)}$ appearing in backpropagation (2)–(3), capturing the effect layers k and above. Note that $A_i^{(k)}$ contains the derivative of the activation of all layers above $k-1$. **top right:** $A_i^{(k)}$ appears as $K_i^{(k)} = A^{(k)} y_i y^T A^{(k)^T}$ in the weight SGD equations. The operator $K_c^{(k)}$ is the average of $K_i^{(k)}$ over all inputs of the same class c. **Bottom:** Distribution of $\delta\rho/\delta N$ on MNIST showing $\mathrm{Var}[\frac{\delta\rho}{\delta N}] \propto \frac{\rho^2}{N^2}$. The fluctuations in the eigenvalues $\delta\lambda_\mu/\delta N$ of the covariance matrix takes a random Gaussian distribution with zero mean and constant variance over added samples N when scaled by N/λ (inset). Averaging this distribution over all eigenvalues confirms that they all have the same λ^2/N^2 variance pattern.

layers $m > k$ in this stage (Fig. 1). We may expand the gradient in a Taylor series to get

$$\frac{\partial g}{\partial \tilde{h}_i^{(n)}} \approx \frac{P_i}{N}\left(\tilde{h}_i^{(n)} - \tilde{y}_i \right), \quad \tilde{y}_i \equiv \log\left(y_i Z_i \right), \tag{6}$$

where $Z_i = \sum_{a=1}^{C} \exp[\tilde{h}_{ia}^{(n)}]$ is the softmax partition function[1] and $P_i \equiv \mathrm{diag}\left(y_i\right) = y_i y_i^T$ is the projection matrix onto the class of y_i. To further simplify (6), we define the "optimal input" $\overline{h}_i^{(0)}$ as the input that would produce exactly the output vector $\tilde{y}_i$

$$\tilde{h}_i^{(n)} = F[h_i^{(0)}], \quad \tilde{y}_i = F[\overline{h}_i^{(0)}] \tag{7}$$

where $F[\cdot]$ summarizes forward propagation of the input through the network. In practice, $\overline{h}_i^{(0)}$ can be obtained via activation maximization [3]. Note that $\overline{h}_i^{(0)}$ is not unique because of the non-convexity of the cost function, as well as weights matrices not being full-rank. Let $\overline{h}_i^{(k)}$ denote the raw output of layer k after propagating $\overline{h}_i^{(0)}$ through the network.

[1] We can replace the zeros in y_i with a small positive $\varepsilon \sim 1/Z_i$ to make $\tilde{y}_i$ well-defined.

We can always find $\overline{h}_i^{(0)}$ such that the activation patterns of $\overline{h}_i^{(k)}$ and $\tilde{h}_i^{(k)}$ are similar (i.e. $f'\left(\overline{h}_i^{(k)}\right) \sim f'\left(\tilde{h}_i^{(k)}\right)$, meaning that $\overline{h}_i^{(k)}$ uses features similar to $\tilde{h}_i^{(k)}$ in each layer). This ensures that $A^{(k)}$ will be the same for $\overline{h}_i^{(k)}$ and $\tilde{h}_i^{(k)}$ [2]. Defining $\Delta h_i^{(k)} \equiv \text{diag}\left(f'\left(\tilde{h}_i^{(k)}\right)\right)\left(\tilde{h}_i^{(k)} - \overline{h}_i^{(k)}\right)$, we can rewrite (6) and substitute in (3) to get

$$\frac{\delta g}{\delta w^{(k)^T}} \approx \frac{1}{N}\sum_{i=1}^{N} h_i^{(k-1)} \Delta h_i^{(k-1)^T} w^{(k)} K_i^{(k+1)}, \quad (8)$$

$$K_i^{(k)} \equiv A^{(k)} y_i y_i^T A^{(k)^T}, \quad (9)$$

where bias-dependent terms exactly cancel. $K_i^{(k)}$ has a structure similar to an auto-encoder restricted to class of y_i. When a DNN learns to classify a dataset it finds high-level features that are characteristic of each class. Many inputs for the same class will share many or some of these high-level features, which translates to the activation pattern $f'\left(\tilde{h}_i^{(k)}\right)$ in (5) being highly correlated for all i belonging to the same class c. This assumption allows us to group inputs for each class c and write (8) as (see [4] for details)

$$\frac{\delta w^{(k)}}{\delta N} \approx -\frac{\varepsilon}{2}\sum_{c=1}^{C} \frac{\delta \rho_c^{(k-1)}}{\delta N} w^{(k)} K_c^{(k+1)}, \quad (10)$$

$$\rho_c^{(k)}(N) \equiv \frac{1}{N}\sum_{i \in c} h_i^{(k)} h_i^{(k)^T} \quad (11)$$

where $K_c^{(k+1)}$ is the mean of $K_i^{(k+1)}$ for all i in class c (Fig. 2 Top Right) and we have defined the "density matrix"[3] $\rho_c^{(k)}$ for each class c. It can be shown, and directly verified from data (Fig. 2 bottom; see [4]), that the fluctuations of the density (and covariance) matrix near convergence follow a multivariate Gaussian

$$\frac{\delta \rho_c^{(k)}}{\delta N} = \mathcal{N}(0,1)\frac{2}{\sqrt{\delta N}}\rho_c^{(k)}/N \quad (12)$$

with $\mathcal{N}(0,1)$ being a standard normal distribution with mean zero and unit standard deviation. Eq. (10) is still hard to solve because $w^{(k)}$ is sandwiched between two other matrices. The presence of $K_c^{(k+1)}$ is generally what makes SGD highly nonlinear, coupling weights of multiple layers. However, notice that before layers $m > k$ converge $K_c^{(k)}/\|K_c^{(k)}\|$ is approximately a projection matrix[4] onto class c. Thus, we can define "class-restricted weights," $w_c^{(k)} \equiv w^{(k)} K_c^{(k+1)}/\|K_c^{(k)}\|$ satisfying $w_{c'}^{(k)} \overline{w}_c^{(k)^{-1}} \approx \delta_{cc'} I$, with $\overline{w}_c^{(k)^{-1}}$ being a pseudo-inverse, found using SVD. We use this to rewrite $\delta w^{(k)}/\delta N$ in terms of $w_c^{(k)}$, but since gradients for lower layers are much larger than higher ones (Fig. 1), we can neglect $\delta\left(K_c^{(k+1)}/\|K_c^{(k)}\|\right)/\delta N$ terms [4]. Multiplying both sides of (10) by $\overline{w}_c^{(k)^{-1}}$ and plugging in (12) into (10), for each class we have a stochastic equation given by

$$\frac{\delta \log_R w_c^{(k)}}{\delta N} = -\varepsilon(N)\rho_c^{(k-1)} \quad (13)$$

where $\varepsilon(N) \equiv \frac{\varepsilon \mathcal{N}(0,1)}{N\sqrt{\delta N}}$, is a random Gaussian noise and the "right logarithm," is formally defined such that $\delta \log_R w_c^{(k)} = \left(\delta w_c^{(k)}\right)\overline{w}_c^{(k)^{-1}}$. Similar to a random walk, we can find the distribution of $\log_R w_c^{(k)}$ becomes a multivariate Gaussian distribution [4] with covariance $\rho_c^{(k-1)^2}/\left\langle \varepsilon(N)^2 \right\rangle$. Thus, $w_c^{(k)}$ is more likely to spread along eigenvectors of $\rho_c^{(k-1)}$ with largest eigenvalues. This means that the class-restricted weights $w_c^{(k)}$ can be found via PCA on $h_i^{(k-1)}$ for all inputs i in class c. The full weight matrix $w^{(k)}$ can be found by combining the $w_c^{(k)}$ for all classes c, as we elaborate below.

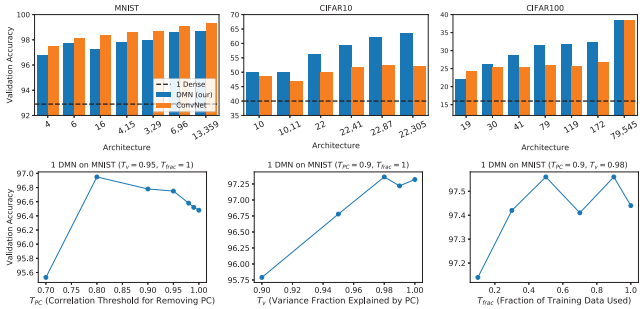

Fig. 3. DMN with one or two layers versus ConvNet: The dashed horizontal line indicates the performance of a network consisting of a single fully connected classification layer, showing the baseline performance. The bar plots have layers of DMN or ConvNet before this classification layer. The structure of the DMN (blue) and the ConvNet (orange) are the same with the same number of filter and layers. The y-axis shows the validation accuracy (percentage) and the x-axis labels show the number of filter ("4" means a single convolutional layer with 4 filters; "4,15" means two layers, first with 4 and second with 15 filters). All experiments have 1 dense classification layer with softmax activation. Each DMN and ConvNet layer is ReLU activated and is followed by a 2×2 maxpooling layer. All filters in DMN and ConvNet have 3×3 receptive fields. In all simulations the correlation threshold to remove duplicate PC was $T_{PC} = 0.9$. The fraction of training data used for DMN was $T_{frac} = 0.3 - 0.5$ for MNIST, $T_{frac} = 0.7$ for CIFAR10, and $T_{frac} = 1.0$ for CIFAR100. **Performance as a function of three DMN parameters:** The three hyperparameters of the DMN, each showing a nontrivial optimal value.

[2] This is a reasonable assumption as we are close to convergence and we can always find an $\overline{h}_i^{(k)}$ which is close enough to $\tilde{h}_i^{(k)}$ so that this is satisfied.

[3] When $h_i^{(k)}$ are mean zero, the density matrix is just the covariance matrix.

[4] This is because, when k is still converging, higher layers $m > k$ evolve much more slowly than layer k, and $w^{(m)}$ are still random matrices.

3. TRAINING LAYERS USING CLASS-BASED PCA

Our results state that a subset of weights $w_c^{(k)}$ is likely to be along principal components (PC) of $h_i^{(k-1)}$ for each class c. Since $wK = wSS^TK$ for any orthogonal S, we are free to choose the basis of the weights for each layer (i.e. the choice of basis is part of the design of the network architecture). We will therefore choose distinct rows of $w^{(k)}$ to be $w_c^{(k)}$, dedicated to each class. If two PCs from two different classes are correlated more than a threshold, T_{PC}, we keep only one of them. Tests with different T_{PC} (Fig. 3 bottom, left) on MNIST show a prominent peak near $T_{PC} \approx 80\%$. Notably, if we do not drop highly correlated PCs across classes (i.e. $T_{PC} \to 1$) the performance drops. We suspect that keeping similar PCs biases the statistics of the data, resulting in worse classification. We choose $b^{(k)} = 0$, as they canceled in (8). Thus, we can construct pretrained neural networks with weights found using the class-based PCA. We will call such a network a "Density Matrix Network" (DMN). Since DMN is based on PCA, we need to choose a threshold $0 < T_v \leq 1$ for how much of the variance of the data we wish to explain (Fig. 3 bottom, middle). We choose one T_v for each layer, using the same T_v for all classes[5] T_v determines the number of filters (or hidden neurons) and the amount of information retained from the input. We find that there is an optimal T_v yielding the best performance. Lastly, DMN can be constructed with a fraction of the training data, since the density (and covariance) matrix can converge with a small amount of data. The fraction T_{frac} of data used to train a DMN (Fig. 3 bottom, right) is an important factor in determining training time. For MNIST we see that for the first layer as low as 30-40% of the training data yields similar to using all of the data. A reasonable fraction may be determined prior by checking if the convergence of ρ_c. the amount of variance explained by PCA for each class (T_v, middle) is another parameter, which also shows a peak around 98% when keeping the other two parameters fixed. As we see keeping all of the variance does *not* result in the best classification accuracy. As we see, all three hyperparameters of DMN have an intuitive, information-theoretic interpretation. Thus, there should be a way to estimate the optimal T_v and T_{PC} based on rate-distortion theory [5] to minimize resource usage while maximizing discrimination among classes.

3.1. Simulations

DMN can be made with virtually any architecture, including convolutional. For the first layer, this is related to the Karhunen-Loeve transform [6] of images, where one breaks an image down into blocks and PCA is performed on the block images. However, in higher layers, since DMN relies on the covariance matrix of the input, overlapping receptive fields result in correlated outputs and, hence, unwanted PCs not arising from covariance of input data [4]. One simple way to avoid correlated outputs is to use a maxpooling layer right after a convolutional DMN. The architectures we used for our experiments consist of one or two convolutional layers with ReLU activation functions, each followed by a maxpooling layer and ending with a classification layer with softmax activation function. We compare the performance of convolutional DMN layer with a similar ConvNet [7] trained using SGD (More experiments with hybrids of DMN and ConvNet, as well Batch Normalization [8] presented in [4]). In DMN, we calculate the Conv layer weights and fix the weights, but we still train the classification layer using SGD. On MNIST (Fig. 3 top, left) DMN (blue) performs within 1% accuracy of ConvNet (orange) almost in all tested cases. On CIFAR10 (Fig. 3 top, center), DMNs consistently outperform the ConvNets, with a very impressive margin of between $5 - 10\%$ in the two layer setting. On CIFAR100 (Fig. 3 top, right), the single layer DMN outperforms ConvNet, while in two layers the performance is equal.

3.2. Time Complexity of DMN vs SGD

The complexity of SGD is due to the matrix products for backpropagation. Training the first layer as DMN and feeding the output into SGD layers removes the need to recalculate the output of the first layer again at every step, and adds a $O\left(d^{(0)}d^{(1)}\right) + O\left({d^{(0)}}^2\right)$ from the PCA. In most cases, especially convolutioanl layers, PCA time complexity is much smaller than recalculating the layer output at every step and using DMN can reduce the training complexity significantly.

4. CONCLUSION

We showed that a hierarchicy of features in data can result in a separation time scales in the convergence of various layers. Lower layers, learning low-level features, will converge at earlier stages. Focusing on the time scale at which a particular layer k is converging, while higher layers have not converged, we found that the layer is likely to converge to a class-based PCA, with some weights (i.e. units in a dense layer or the filters in a convolutional layer) learning PCs of each class. This observations allows us to train networks by performing PCA on inputs for each class, resulting in a fast and efficient training at least for lower layers. Using this method, we were able to produce pretrained convolutional layers for MNIST, CIFAR10 and CIFAR100 which perform on par or superior to convolutional layers trained via SGD. The hyperparameters of this method are related to the variance explained in the PCA and how informative and discriminative the weights are. Tuning hyperparameters of a specific layer can also be done very efficiently, as the PCA needs to be performed only once. Other recent work [9] using information theory has also suggested that SVD and PCA can be used as universally optimal guesses for weights in deep neural networks, supporting our results.

[5] Setting $T_v = 1$ means all eigenvectors of $\rho_c^{(k)}$ are kept, making DMN a linear transformation. If no activation function is used after DMN, $T_v = 1$ yields no improvement to the classification compared to when the layer is absent On the other hand, very small T_v also decreases the performance.

5. REFERENCES

[1] Matus Telgarsky. Benefits of depth in neural networks. *arXiv preprint arXiv:1602.04485*, 2016.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[3] Aravindh Mahendran and Andrea Vedaldi. Visualizing deep convolutional neural networks using natural pre-images. *International Journal of Computer Vision*, 120(3):233–255, 2016.

[4] Nima Dehmamy, Neda Rohani, and Aggelos Katsaggelos. Separation of time scales and direct computation of weights in deep neural networks. *arXiv preprint arXiv:1703.04757*, 2017.

[5] Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[6] M Loève. Graduate texts in mathematics. *Probability Theory I,*, pages 159–160, 1977.

[7] Yann LeCun and Yoshua Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[8] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[9] Shao-Lun Huang, Anuran Makur, Lizhong Zheng, and Gregory W Wornell. An information-theoretic approach to universal feature selection in high-dimensional inference. In *Information Theory (ISIT), 2017 IEEE International Symposium on*, pages 1336–1340. IEEE, 2017.