Autoencoder Neural Networks With Nested Alternating Minimization

Eyal Gur

August 12, 2024

Let $\mathcal{X}$ represent a dataset of N unlabeled points $\mathbf{x}^i \in \mathbb{R}^n$. Define $\mathcal{E}_\theta \colon \mathbb{R}^n \to \mathbb{R}^m$ and $\mathcal{D}_\phi \colon \mathbb{R}^m \to \mathbb{R}^n$ as the encoder and decoder functions, respectively, where the decoder is parameterized by the set of weights ϕ and the encoder by θ. We then consider the regularized autoencoder minimization problem

$$\min_{\phi,\theta} \ \mathcal{A}(\phi,\theta) \equiv \sum_{\mathbf{x}\in\mathcal{X}} \left(\left\| \mathbf{x} - \mathcal{D}_\phi\left(\mathcal{E}_\theta(\mathbf{x})\right) \right\|^2 + \mathcal{R}(\phi,\theta) \right),$$

where $\mathcal{R}(\phi,\theta)$ is some regularizing function.

This structure calls for the Nested Alternating Minimization (NAM) scheme, where the decoder and encoder – the partial functions $\phi \mapsto \mathcal{A}(\phi,\theta)$ and $\theta \mapsto \mathcal{A}(\phi,\theta)$, respectively – are minimized in an alternating fashion.

1 NAM with ADAM

We assume that the encoder produces sparse encodings by applying ℓ_1 regularization to the encoded representations, meaning that we enforce sparsity in the latent space of the encodings. Mathematically, $\mathcal{R}(\phi,\theta) = \lambda \sum_{\mathbf{x}\in\mathcal{X}} \left\| \mathcal{E}_\theta(\mathbf{x}) \right\|_1$ for $\lambda > 0$ (notice that $\mathcal{R}$ depends only on the parameterization of the encoder).

We apply the NAM scheme on the autoencoder function $\mathcal{A}(\phi,\theta)$, where both the decoder and encoder are minimized using iterations of the ADAM optimizer. Specifically, when minimizing the decoder, only its trainable weights are updated while the weights of the encoder remain fixed, and the same applies to the encoder during its updates. We compare the NAM scheme to the conventional ADAM optimizer applied to the autoencoder (referred to as the Joint Model in the experiments below), where all trainable weights are updated in every iteration.

The NAM Model can be seen as a generalization of the Joint Model, offering the advantage of separately tuning hyperparameters for the decoder and encoder. By examining the

structure of the autoencoder function, it is evident that the optimization landscapes of the decoder and encoder differ. The NAM approach allows us to set different hyperparameters, such as step size, epochs, batch sizes, number of iterations, etc., for the decoder and encoder independently, possibly enabling a more tailored and effective optimization process.

The implementation of the NAM Model and the Joint Model using PyTorch on the CIFAR10 dataset is given in this Google Colab notebook. Both models are trained on a simple convolutional neural network, where the encoder consists of three layers that transform from input size $3\times32\times32$ to $48\times4\times4$ with ReLU activations. The decoder consists of three layers that transform the encoded output from size $48 \times 4 \times 4$ back to the input size $3 \times 32 \times 32$ with ReLU activations and a sigmoid at the last layer.

Full network architecture, dataset specifications, and the setting of hyperparemeters, are provided in the notebook.

1.1 Results

In the preliminary results presented below, we observe that the Joint Model achieves a lower loss compared to the NAM Model. However, other critical factors reveal a trade-off between the two approaches. The NAM Model demonstrates *significant* advantages in computational efficiency, speed, and compression effectiveness. These criteria suggest that the NAM Model may be a more balanced choice, depending on the specific requirements of the task.

We conduct 10 trials, each with different randomly initialized tensors. All results presented below are averaged across these trials. We use 12 epochs with a batch size of 125, resulting in 400 iterations per epoch, given that the CIFAR10 training set contains 50000 images. The regularization constant is set to $\lambda = 10^{-8}$, noting that this value is significantly influenced by the ℓ_1 norm of the images.

For the Joint Model, we use a learning rate of 0.001 (the default rate). For the NAM Model, we set a learning rate of 0.01 for the decoder and 0.001 for the encoder. Both models are configured to run the maximum number of iterations (i.e., 400 iterations per epoch), although in the NAM Model, this can be adjusted separately for the encoder and decoder. Additionally, in the NAM Model, updates for the decoder and encoder alternate with each epoch, though this switching can occur after any number of iterations within an epoch.

Loss. In Figure 1, we present the average training loss across the 10 trials. The results show that the Joint Model performs slightly better than the NAM Model. Regarding test loss, the average for the Joint Model is 0.0186, which is better than the average of the NAM Model of 0.0212.

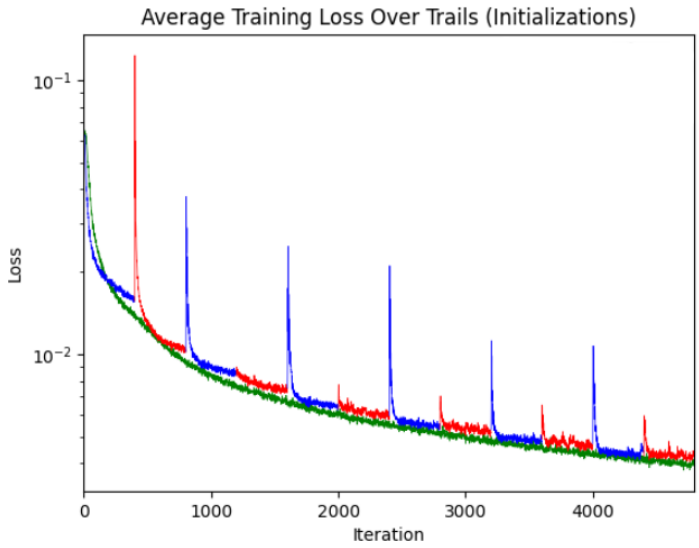


Figure 1: Average training set loss: the Joint Model is shown in green, while the NAM Model is represented by alternating blue (decoder updates) and red (encoder updates).

Cost and runtime. Although the Joint Model demonstrates better performance in terms of loss, these results should be interpreted with caution due to the higher training cost associated with the Joint Model. It is important to note that in each iteration, the NAM Model updates only the weights of either the decoder or the encoder, whereas the Joint Model updates all weights simultaneously. As a result, the Joint Model updates 2 times *more* weights per epoch compared to the NAM Model, indicating that the computational training cost per epoch for the Joint Model is twice that of the NAM Model.

In terms of runtime, the NAM Model is approximately 15% *faster* per epoch than the Joint Model (exact figures are provided in the notebook). This suggests that the NAM Model can be trained for *more* iterations within a *shorter* overall runtime compared to the Joint Model, indicating that the loss results discussed earlier do not fully account for these differences in cost and runtime.

Compression. Another important aspect to consider is how effectively each model encodes (compresses) the images, as this is a key objective of this autoencoder network.

The average ℓ_1 norm of the test images is 1415.94. After encoding (compression), the average ℓ_1 norm of these images is 3.59 for the Joint Model and 0.55 for the NAM Model. This indicates that the NAM Model achieves 6.57 times *more* compression than the Joint Model (exact figures are provided in the notebook).

It is also important to note that with larger regularization constants $\lambda > 0$, the Joint Model diverges, resulting in all images being compressed to 0 (making them impossible to decode), whereas the NAM Model continues to converge successfully.

Conclusion. In summary, while the Joint Model achieves a lower loss, it comes with significant trade-offs. The NAM Model, on the other hand, updates fewer weights per iteration, making it more computationally efficient and cost-effective. Additionally, the NAM Model is faster per epoch and provides superior image compression, achieving a higher level of sparsity in the encoded representations. These advantages make the NAM Model a more efficient and potentially more effective choice, especially when considering the overall training cost, speed, and compression performance.

2 FISTA for Autoencoders

For convenience, we define the two functions $f(\phi,\theta) \equiv \sum_{\mathbf{x}\in\mathcal{X}} \|\mathbf{x} - \mathcal{D}_\phi(\mathcal{E}_\theta(\mathbf{x}))\|^2$ and $g(\theta) \equiv \sum_{\mathbf{x}\in\mathcal{X}} \|\mathcal{E}_\theta(\mathbf{x})\|_1$, thus the autoencoder can be written as

$$\min_{\phi,\theta} \; f(\phi,\theta) + \lambda g(\theta).$$

In this document, we assume that both decoder and encoder are parametarized with differentiable activation functions (for instance, sigmoid). A somewhat weaker suitable assumption is that the function f is differentiable at any data point $\mathbf{x} \in \mathcal{X}$.

Developing the ANAMO Algorithm

The structure of the autoencoder as a minimization problem that possesses two blocks of variables ϕ and θ, naturally calls for my *Nested Alternating Minimization* (NAM) scheme, which is an optimization scheme developed and analyzed in my PhD thesis. By applying this scheme, we solve the minimization problem iteratively in the following alternating fashion: first, we fix ϕ and minimize the partial function $\theta \mapsto \mathcal{A}(\phi,\theta)$ using some nested algorithm. Then, we fix θ with its updated value and minimize the partial function $\phi \mapsto \mathcal{A}(\phi,\theta)$ using some nested algorithm. The process is repeated until a stopping criterion is satisfied.

Focusing on the partial function with respect to the encoder, that is $\theta \mapsto \mathcal{A}(\phi,\theta)$, this function is the sum of the differentiable partial function $\theta \mapsto f(\phi,\theta)$, and the non-differentiable regularization function $g(\theta)$. Hence, the minimization problem with respect to the encoder can be minimized, for example, using the *FISTA* algorithm (a variant of Accelerated Gradient for the non-differentiable composite setting).

As for the decoder, we notice that the partial function $\phi \mapsto \mathcal{A}(\phi,\theta)$, is differentiable as it coincides with $\phi \mapsto f(\phi,\theta)$. Therefore, the minimization problem with respect to the decoder can be minimized using inner iterations of the Accelerated Gradient method (or its stochastic variants).

My overall suggested algorithm, called *Autoencoder NAM Optimizer* (ANAMO), is formulated now in Algorithm 1. The gradients in this algorithm can be calculated, for example, with backpropagation.

Theoretical Justifications for ANAMO

There are several theoretical justifications for ANAMO (that is, applying NAM with FISTA for autoencoders):

1. Convergence rate: the FISTA method is known to enjoy a fast linear convergence rate (that is, $\mathcal{O}(1/n^2)$ where n is the number of iterations). This is compared to other

Algorithm 1 Autoencoder NAM Optimizer (ANAMO)

Initialization: Random (ϕ^0, θ^0), learning rates $\{e_j\}_{j\geq 0}$ and $\{d_j\}_{j\geq 0}$, a sequence $\{s_j\}_{j\geq 0}$ such that $s_{j+1} = \left(1+\sqrt{1+4s_j^2}\right)/2$ for $s_0 = 1$, and mini-batches $\{\mathcal{X}_k\}_{k\geq 0} \subseteq \mathcal{X}$.
for $k \geq 0$ **do**
  Update the encoder for mini-batch k:
  Set $\tilde{\theta}^0 = \mathbf{y}^0 = \theta^k$.
  for $j = 0, 1, \ldots, J-1$ **do**
    Update $\tilde{\theta}^{j+1} = \text{argmin}_{\mathbf{y}}\left\{\lambda g(\mathbf{y}) + \frac{1}{2}\left\|\mathbf{y} - \mathbf{y}^j + e_j \nabla_\theta f\left(\phi^k, \mathbf{y}^j\right)\right\|^2\right\}$.
    Update $\mathbf{y}^{j+1} = \tilde{\theta}^{j+1} + \frac{s_j - 1}{s_{j+1}}\left(\tilde{\theta}^{j+1} - \tilde{\theta}^j\right)$.
  end for
  Set $\theta^{k+1} = \tilde{\theta}^J$.
  Update the decoder for mini-batch k:
  Set $\tilde{\phi}^0 = \mathbf{y}^0 = \phi^k$.
  for $j = 0, 1, \ldots, J-1$ **do**
    Update $\tilde{\phi}^{j+1} = \mathbf{y}^j - d_j \nabla_\phi f\left(\mathbf{y}, \theta^{k+1}\right)$.
    Update $\mathbf{y}^{j+1} = \tilde{\phi}^{j+1} + \frac{s_j - 1}{s_{j+1}}\left(\tilde{\phi}^{j+1} - \tilde{\phi}^j\right)$.
  end for
  Set $\phi^{k+1} = \tilde{\phi}^J$.
end for

gradient-based methods, such as Gradient Descent and some of its stochastic variants, that only have sub-linear convergence rate (that is, $\mathcal{O}(1/n)$).

2. Global convergence: in my thesis, I formulated and proved convergence guarantees of the NAM scheme to a first-order optimal solution in some non-convex and non-differentiable cases. These results hold even though that FISTA, as a stand-alone algorithm, do not possesses such theoretical convergence guarantees. This implies that under certain settings, ANAMO may converge to first-order optimal solutions of the minimization problem.

3. Adjustable optimization process for encoder and decoder: the optimization landscape need not be similar for the encoder and decoder, and in fact it can be quite different as the encoder sets the input for the decoder. Therefore, it is favorable to allow flexibility in the optimization process of the decoder and encoder, which can be obtained by using ANAMO and *not* by other gradient-based methods:

   (a) The step-sizes (which can also be adaptable) can be set *differently* for the updates of the encoder and decoder.

(b) One can apply a *different* number of iterations to update the weights of the encoder and decoder.

(c) Even a different type of a nested optimization algorithm can be set for the encoder and the decoder.

4. Regularization: a regularization term can be easily incorporated to the autoencoder (though the optimization process need *not* be simple), as FISTA can handle such terms.

Comparison of ANAMO and ADAM

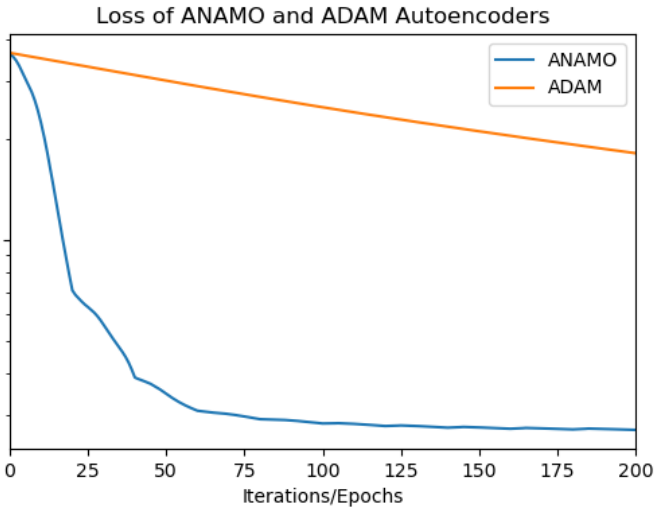
For the sake of a simple comparison, we compare the two methods for a single-layered encoder and decoder, both with a sigmoid activation function. The NumPy implementation of ANAMO for this specific instance (including backpropagation) is attached to this document.

We synthesized 100 random gray-scale training images of size 10×10 pixels, each with a whiter left side compared to its blacker right side. The size of the hidden layer was set to 10. For both algorithms, we set the same random initial weights for the encoder and decoder.

For ANAMO, we set 5 (inner) update iterations for the encoder and 15 for the decoder. In addition, we set 10 outer iterations. The learning rates for both were set to n, and the regularization constant was set to $\lambda = 0.1$.

For ADAM, we used its default hyper-parameters. The number of epochs was set to $10 \times (5 + 15) = 200$ (just for the sake of a simpler comparison, as this number includes more total iterations compared to ANAMO).

1. In the results below, we compared the value of the loss (as formulated at the beginning of this document) along the iterations/epochs.



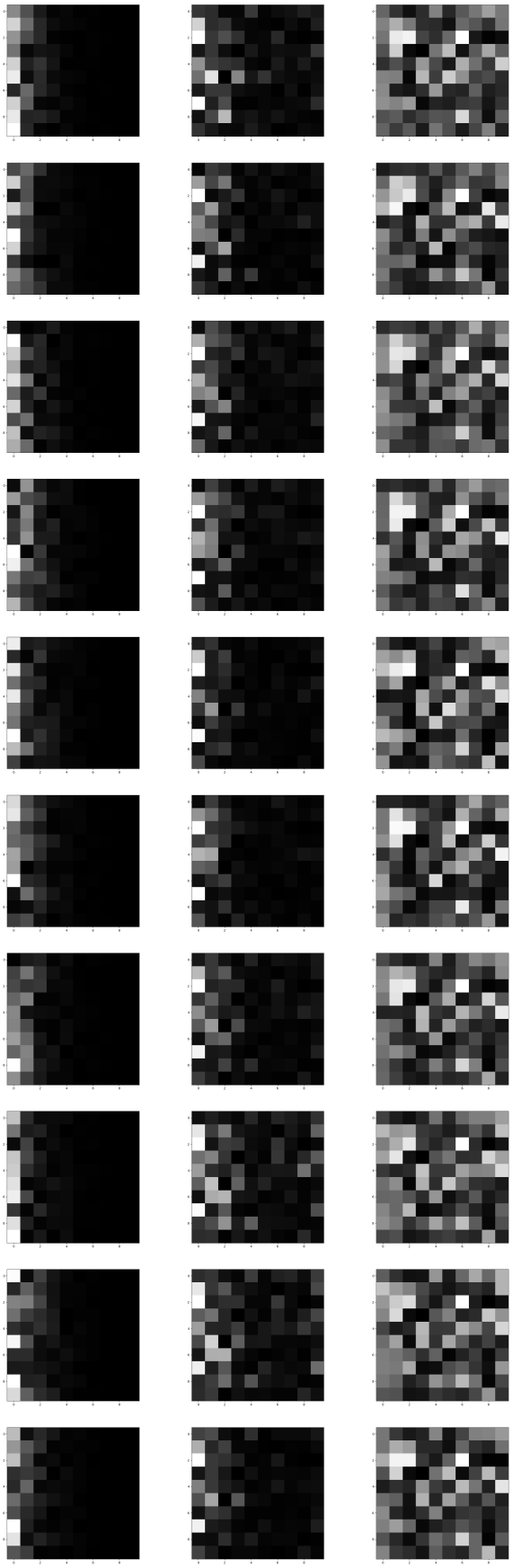
2. In addition, we synthesized 10 random test images and compared the average reconstruction bias for both algorithms. The formula for the average bias is

$$\frac{1}{M}\sum_{i=1}^{M}\left\|\mathbf{x}^i_{\text{real}}-\mathbf{x}^i_{\text{pred}}\right\|,$$

where M is the size of the test-set, $\mathbf{x}^i_{\text{real}}$ is a test image, and $\mathbf{x}^i_{\text{pred}}$ is its prediction by one of the algorithms.

For the test set, ANAMO obtained a value of 1.85 and ADAM of 3.70.

3. Last, we plot the test images (left column), along with the predictions made by ANAMO (middle column), and by ADAM (right column).



With the results above, there are still some questions that need to be addressed in the future regarding ANAMO. For example: how the implementation of ANAMO can be extended, theoretically and numerically, to more complex activation functions? How can we calculate or approximate the prox term (see step 6 in ANAMO) in more complex settings? How can we estimate suitable step-sizes for the encoder and decoder? What is a suitable number of inner iterations that will guarantee good results but will not increase the overall running time?

Binary Encoder

In this case, since the decoder is left unchanged, we can still update the decoder in the same way as previously discussed (see steps 10–16 in ANAMO). However, since $\mathcal{E}_\theta(\mathbf{x})$, $\mathbf{x} \in \mathcal{X}$, is now a binary vector, there are no simple means to calculate the involved prox term for the encoder with its regularization term (see step 6 in ANAMO). We now propose three ways to try and tackle this issue, and to approximate a suitable encoder for the regularized problem.

One way, is to apply inner iterations of some nested algorithm (for example, sub-gradient descent) to approximate the prox term. This is formulated in Algorithm 2. However, a convenient way for finding sub-gradients for the partial function with respect to the encoder must be obtained.

Algorithm 2 ANAMO with Binary Encoder – V1

1: **Initialization:** Random (ϕ^0, θ^0), learning rates $\{e_j\}_{j\geq 0}$ and $\{d_j\}_{j\geq 0}$, a sequence $\{s_j\}_{j\geq 0}$ such that $s_{j+1} = \left(1+\sqrt{1+4s_j^2}\right)/2$ for $s_0 = 1$, and mini-batches $\{\mathcal{X}_k\}_{k\geq 0} \subseteq \mathcal{X}$.
2: **for** $k \geq 0$ **do**
3: *Update the encoder for mini-batch k:*
4: Set $\tilde{\theta}^0 = \mathbf{y}^0 = \theta^k$.
5: **for** $j = 0, 1, \ldots, J-1$ **do**
6: Apply iterations of sub-gradient descent to approximate a solution $\tilde{\theta}^{j+1}$ to step 6 of ANAMO.
7: Update $\mathbf{y}^{j+1} = \tilde{\theta}^{j+1} + \frac{s_j - 1}{s_{j+1}}\left(\tilde{\theta}^{j+1} - \tilde{\theta}^j\right)$.
8: **end for**
9: Set $\theta^{k+1} = \tilde{\theta}^J$.
10: *Update the decoder for mini-batch k:* repeat steps 11–16 of ANAMO
11: **end for**

Another way to tackle this problem is to apply some smooth approximation function of the $\{0,1\}$-step activation function (for example, the function $3\left(\frac{x}{\epsilon}\right)^2 - 2\left(\frac{x}{\epsilon}\right)^3$ is a suitable smooth choice, though it introduces $\epsilon > 0$ as an additional hyper-parameter). With this

smooth approximating activation function, we can use ANAMO under the setting discussed in Algorithm 1.

An additional way, is to notice that we are trying to find a binary representation $\mathbf{h}^i$ for each data point $\mathbf{x}^i \in \mathcal{X}$ in the dataset, $i = 1, 2, \ldots, N$. A possible way to find suitable binary representations is to solve the simplified regularized problem

$$\min_{\phi, \tilde{\mathbf{h}}} \mathcal{A}\left(\phi, \tilde{\mathbf{h}}\right) \equiv \sum_{\mathbf{x} \in \mathcal{X}} \left(\left\| \mathbf{x} - \mathcal{D}_\phi\left(\tilde{\mathbf{h}}^i\right) \right\|^2 + \lambda \left\| \tilde{\mathbf{h}}^i \right\|_1 \right),$$

which can be solved, for instance, using ANAMO (or any other suitable method, such as sub-gradient descent). Once the optimization process is completed, and in order to obtain the the final binary encodings, we update $\mathbf{h}^i \equiv \text{step}\left(\tilde{\mathbf{h}}^i\right)$ for all $i = 1, 2, \ldots, N$, where step is the $\{0, 1\}$-step function. Finally, in order to estimate the encoder $\mathcal{E}_\theta$, we solve the set of equations $\mathcal{E}_\theta\left(\mathbf{x}^i\right) = \mathbf{h}^i$ for $i = 1, 2, \ldots, N$, which can be performed, for example, by means of regression. This algorithm is given in Algorithm 3.

Algorithm 3 ANAMO with Binary Encoder - V3

1: **Initialization:** Random $\left(\phi^0, \tilde{\mathbf{h}}^0\right)$, learning rates $\{e_j\}_{j \geq 0}$ and $\{d_j\}_{j \geq 0}$, a sequence $\{s_j\}_{j \geq 0}$ such that $s_{j+1} = \left(1 + \sqrt{1 + 4s_j^2}\right)/2$ for $s_0 = 1$, and mini-batches $\{\mathcal{X}_k\}_{k \geq 0} \subseteq \mathcal{X}$.
2: **for** $k \geq 0$ **do**
3: *Update the representations* $\tilde{\mathbf{h}}$ *for mini-batch* k*:* repeat steps 4–9 of ANAMO, where $\tilde{\mathbf{h}}$ is replacing θ.
4: *Update the decoder for mini-batch* k*:* repeat steps 11–16 of ANAMO.
5: **end for**
6: Update $\mathbf{h} \equiv \text{step}\left(\tilde{\mathbf{h}}\right)$.
7: Approximate the encoder $\mathcal{E}_\theta$ by solving the system of equations $\mathcal{E}_\theta\left(\mathbf{X}\right) \approx \mathbf{h}$.