A Probe parameterization

We parameterized the probe functions with a single layer MLP. When masking input tokens, 'votes' are computed as $v_i^{(\ell)} = g_\phi^{(\ell)}(h_i^{(\ell)})$ where

$$\gamma_i^{(\ell)} = \xi \cdot \tanh\left(\mathrm{NN}^{(\ell)}([x_i; h_i^{(\ell)}])\right) + b^{(\ell)} \ , \quad (4)$$

$$v_i^{(\ell)} \sim \mathrm{HardConcrete}(v_i^{(\ell)}; \tau, \gamma_i^{(\ell)}, l, r) \ , \quad (5)$$

where $\xi = 10, \tau = 0.2, l = -0.2, r = 1.0$ are fixed hyperparameters. See Appendix B for details about the Hard Concrete distribution including its parameterization. NN are feed-forward neural networks with architecture $[H/4, \tanh, 1]$ where H is the BERT hidden size, bs are learnable biases. We use the same functional form to compute $z^{(\ell)}$ (masking hidden states) but x_i omitted from the input of the feed-forward NN. For the input probe the output of the last projection (but not the bias) is constrained to be $\in (-\xi, \xi)$ for numerical stability. We initialized the bias of the last FFNN layer to 5 to start with high probability of keeping states (fundamental for good convergence as the initialized DIFFMASK has not learned what to mask yet).

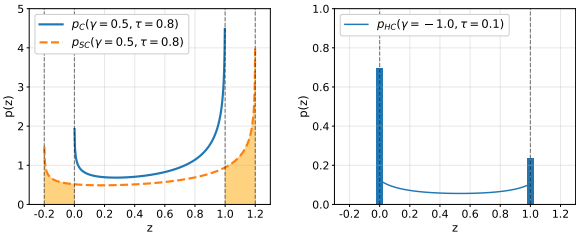
B The Hard Concrete distribution

The Hard Concrete distribution, assigns density to continuous outcomes in the open interval $(0, 1)$ and non-zero mass to exactly 0 and exactly 1. A particularly appealing property of this distribution is that sampling can be done via a differentiable reparameterization (Rezende et al., 2014; Kingma and Welling, 2014). In this way, the $\mathcal{L}_0$ loss in Equation 1 becomes an expectation

$$\mathcal{L}_0(\phi, b|x) = \sum_{i=1}^{N} \mathbb{E}_{p_\phi(z_i|x)}\left[z_i \neq 0\right] \ . \quad (6)$$

whose gradient can be estimated via Monte Carlo sampling without the need for REINFORCE and without introducing biases. We did modify the original Hard Concrete, though only so slightly, in a way that it gives support to samples in the half-open interval $[0, 1)$, that is, with non-zero mass only at 0. That is because we need only distinguish 0 from non-zero, and the value 1 is not particularly important.[7]

[7] Only a true 0 is guaranteed to completely mask an input out, while any non-zero value, however small, may leak some amount of information.



(a) (b)

Figure 9: Binary Concrete distributions: (a) a Concrete p_C and its stretched version p_{SC}; (b) a rectified and stretched (Hard) Concrete p_{HC}.

The distribution A stretched and rectified Binary Concrete (also known as Hard Concrete) distribution is obtained applying an affine transformation to the Binary Concrete distribution (Maddison et al., 2017; Jang et al., 2017) and rectifying its samples in the interval $[0, 1]$ (see Figure 9). A Binary Concrete is defined over the open interval $(0, 1)$ (p_C in Figure 9a) and it is parameterised by a location parameter $\gamma \in \mathbb{R}$ and temperature parameter $\tau \in \mathbb{R}_{>0}$. The location acts as a logit and it controls the probability mass skewing the distribution towards 0 in case of negative location and towards 1 in case of positive location. The temperature parameter controls the concentration of the distribution. The Binary Concrete is then stretched with an affine transformation extending its support to (l, r) with $l \leq 0$ and $r \geq 1$ (p_{SC} in Figure 9a). Finally, we obtain a Hard Concrete distribution rectifying samples in the interval $[0, 1]$. This corresponds to collapsing the probability mass over the interval $(l, 0]$ to 0, and the mass over the interval $[1, r)$ to 1 (p_{HC} in Figure 9b). This induces a distribution over the close interval $[0, 1]$ with non-zero mass at 0 and 1. Samples are obtained using

$$\begin{aligned} s &= \sigma\left(\left(\log u - \log(1-u) + \gamma\right)/\tau\right) \ , \\ z &= \min\left(1, \max\left(0, s \cdot (l - r) + r\right)\right) \ , \end{aligned} \quad (7)$$

where σ is the Sigmoid function $\sigma(x) = (1 + e^{-x})^{-1}$ and $u \sim \mathcal{U}(0, 1)$. We point to the Appendix B of Louizos et al. (2018) for more information about the density of the resulting distribution and its cumulative density function.

Latent rationales There is a stream of work on learning interpretable models by means of extracting latent rationales (Lei et al., 2016; Bastings et al., 2019). Some of the techniques underlying DIFFMASK are related to that line of work, but overall we approach very different problems. Lei

et al. (2016) use REINFORCE to minimize a downstream loss computed on masked inputs, where the masks are binary and latent. They employ L_0 regularization to solve the task while conditioning only on small subsets of the input regarded as a *rationale* for the prediction. To the same end, Bastings et al. (2019) minimize downstream loss subject to constraints on expected L_0 using a variant of the sparse relaxation of Louizos et al. (2018). In sum, they employ stochastic masks to learn an interpretable model which they learn by minimizing a downstream loss subject to constraints on L_0, we employ stochastic masks to interpret an existing model and for that we minimize L_0 subject to constraints on that model's downstream performance.

C Hyperparameters

C.1 Toy task

Data We generate sequences of varying length (up to 10 digits long) sampling each element independently: with 50% probability, we draw uniformly n or m and, with 50% probability, we draw uniformly from the remaining digits. We generate 10k data-points, keeping 10% of them for validation. The space of input sequences is $>10^{10}$. Thus, a model that solves the task cannot simply memorize the training set.

Model The precise model formulation is the following: given a query $q = \langle n, m \rangle$ and an input $x = \langle x_1, \dots x_t \rangle$, they are embedded as

$$\begin{aligned} n' &= \mathrm{Emb}_q(n) \,, \\ m' &= \mathrm{Emb}_q(m) \,, \\ x'_i &= \mathrm{Emb}_x(x_i) \qquad \forall i \in 1 \dots t \,, \end{aligned} \tag{8}$$

where Emb_q and Emb_x are embedding layers of dimensionality 64. The prediction is computed as

$$\begin{aligned} h_i^{(1)} &= \mathrm{FFNN}([n'; m'; x'_i]) \quad \forall i \in 1 \dots t \,, \\ h_0^{(2)} &= [0 \dots 0]^\top \,, \\ h_i^{(2)} &= \mathrm{GRU}(h_i^{(1)}, h_{i-1}^{(2)}) \qquad \forall i \in 1 \dots t \,, \\ y &= w^\top h_t^{(2)} + b \,, \end{aligned} \tag{9}$$

where $[\cdot;\cdot]$ denotes concatenation, FFNN is a feed-forward neural network with architecture $[64 \times 3, \tanh, 2]$, GRU is a Gated Recurrent Network (Cho et al., 2014) with hidden size of 64, and $w \in \mathbb{R}^{64}$, $b \in \mathbb{R}$ are the weight and bias parameter of the final classifier respectively.

Model	**Value**
Type	$\mathrm{BERT}_{\mathrm{BASE}}$ (uncased)
Layers	12
Hidden units	768
Pre-trained masking	standard
Optimizer	Adam*
Learning rate	$3 \cdot 10^{-5}$
Train epochs	50
Batch size	64
DIFFMASK	**Value**
Optimizer	Lookahead RMSprop**
Learning rate ϕ, b	$3 \cdot 10^{-4}$
Learning rate λ	$1 \cdot 10^{-1}$
Train epochs	100
Batch size	64
Constrain	$\mathrm{D}_{\mathrm{KL}}[y \| \hat{y}] < 0.5$

Table 3: Hyperparameters for the sentiment classification experiment. Optimizers: * Kingma and Ba (2015), ** Tieleman and Hinton (2012); Zhang et al. (2019).

Attribution methods Integrated gradient attribution (Sundararajan et al., 2017) is computed with 500 steps. Attribution of Schulz et al. (2020) is computed at token level with $\beta = 10/k$ where k is the token embedding size. We optimized using the RMSprop (Tieleman and Hinton, 2012) with learning rate 10^{-1} for 500 steps. Attribution of Guan et al. (2019) is computed at token level with $\lambda = 10^{-4}$ using RMSprop with learning rate 10^{-1} for 500 steps. Our DIFFMASK is optimized for 100 epochs using Lookahead RMSprop (Tieleman and Hinton, 2012; Zhang et al., 2019) with learning rate 10^{-2} for ϕ, b and 10^{-1} for α. For these attribution methods we used our own re-implementation.

C.2 Sentiment Classification

Data We used the Stanford Sentiment Treebank (SST; Socher et al., 2013) available here[8]. We pre-processed the data as in Bastings et al. (2019). Training and validation sets contain 8544 and 1101 sentences respectively.

Model For the sentiment classification experiment we downloaded[9] a pre-trained model from

[8] https://nlp.stanford.edu/sentiment/trainDevTestTrees_PTB.zip

[9] https://huggingface.co/transformers/pretrained_models.html

Model	Value
Type	$\text{BERT}_{\text{LARGE}}$ (uncased)
Layers	24
Hidden units	1024
Pre-trained masking	whole-word
Optimizer	Adam*
Learning rate	$3 \cdot 10^{-5}$
Train epochs	2
Batch size	24
DIFFMASK	**Value**
Optimizer	Lookahead RMSprop**
Learning rate ϕ, b	$3 \cdot 10^{-4}$
Learning rate λ	$1 \cdot 10^{-1}$
Epochs (inputs)	1 (per layer)
Epochs (hidden)	4
Batch size	8
Constrain	$\text{D}_{\text{KL}}[y \Vert \hat{y}] < 1$

Table 4: Hyperparameters for the question answering experiment. Optimizers: * Kingma and Ba (2015), ** Tieleman and Hinton (2012); Zhang et al. (2019).

the Huggingface implementation[10] of Wolf et al. (2019), and we fined-tuned on the SST dataset. We report hyperparameters used for training the model and our DIFFMASK in Table 3.

C.3 Question Answering

Data We used the Stanford Question Answering Dataset (SQUAD V1.1; Rajpurkar et al., 2016) available here[11]. Pre-processing excluded QA pairs with more than 384 BPE tokens to avoid memory issues. After this we end up having 86706 training instances and 10387 validation instances.

Model For the question answering experiment we downloaded [9] an already fine-tuned model from the Huggingface implementation[10] of Wolf et al. (2019) We report hyperparameters used by them for training the original model and the ones used for our DIFFMASK in Table 4.

D Additional plots and results

In Figure 10 we show an overview of the variant of DIFFMASK to analyze the hidden states of a model (see Figure 1 to compare the two versions).

[10] https://github.com/huggingface/transformers

[11] https://rajpurkar.github.io/SQuAD-explorer

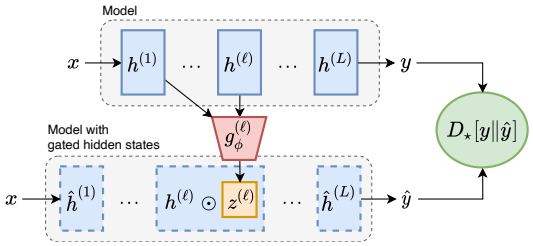


Figure 10: DIFFMASK for hidden states: states up to layer ℓ from a model (top) are fed to a classifier g that predicts a mask $z^{(\ell)}$. We use this to mask the ℓih hidden state and re-compute the forward pass from that point on (bottom). The classifier g is trained to mask the hidden state as much as possible without changing the output (minimizing a divergence $\text{D}_\star$).

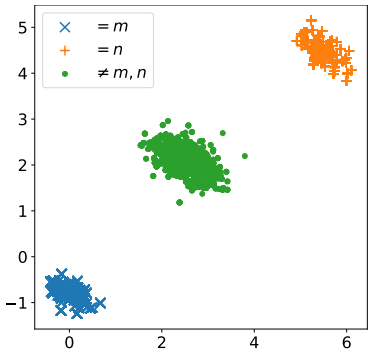


Figure 11: Hidden state values for the two-neuron toy task. Clusters of whether the input digit is equal to the first or second position in the query ($= n$ or $= m$ respectively) or not at all ($\neq n, m$) are completely linear separable.

D.1 Toy task

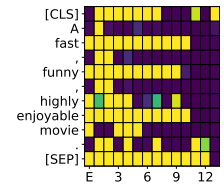
In Figure 11 we show the distribution of hidden states in the toy task where we highlight whether they belong to a state corresponding to n, m or neither of them.

D.2 Sentiment Classification

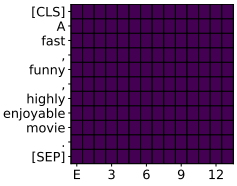
In Figure 13 we show an additional comparison example between attribution method for hidden layers w.r.t the predicted label.

D.2.1 Ablation

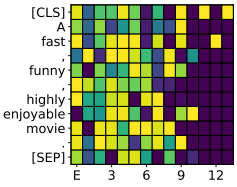
As argued in the introduction and shown on the toy task, many popular methods (e.g., erasure and its approximations) are over-aggressive in discarding inputs and hidden units. Amortization is a fundamental component of DIFFMASK and is aimed at addressing this issue. In Figure 12 we show how our method behaves when ablating amortization and thus optimizing on a single example instead. Noticeable, our method converges to masking out



(a) Masking hidden states with amortization.



(b) Masking hidden states without amortization.



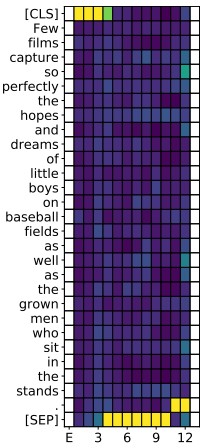
(c) Masking hidden states without amortization and without baseline.

Figure 12: Sentiment classification: ablation study on amortization and baseline.

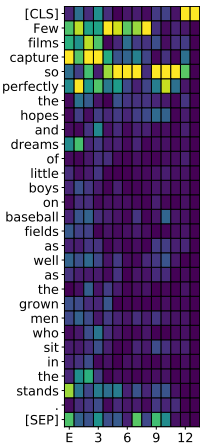
all hidden states at any layer (Figure 12b). This happens as it learns an *ad hoc* baseline just for that example. When we ablate both amortization and baseline learning (Figure 12c), the method struggles to uncover any meaningful patterns. This highlights how both core components of our method are needed in combination with each other.

D.3 Question Answering

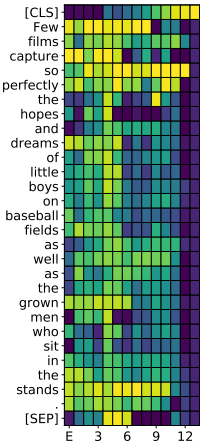
In Figure 14 we report statistics on the average number of layers that predict to keep input tokens aggregating by POS tag. We report additional two examples of expectation predicted by DIFFMASK in Figure 15.



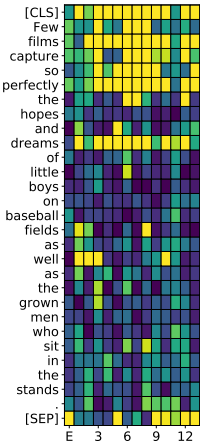
(a) Attention.



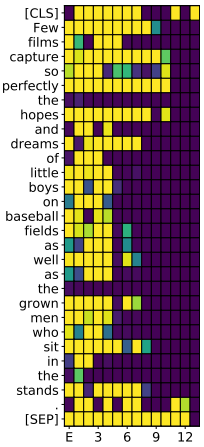
(b) Sundararajan et al. (2017).



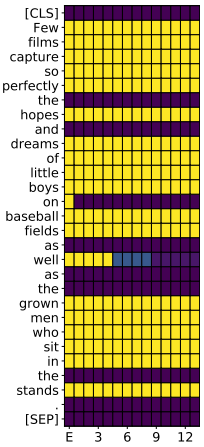
(c) Schulz et al. (2020).



(d) Guan et al. (2019).

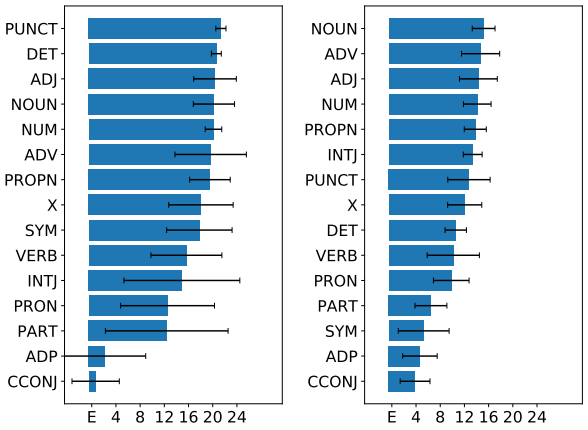


(e) DIFFMASK.



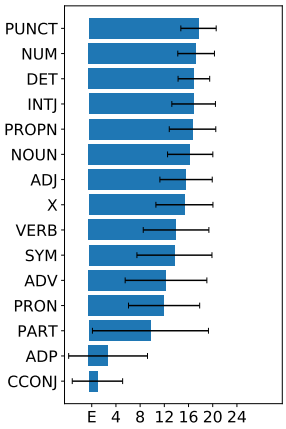
(f) DIFFMASK on input.

Figure 13: Sentiment classification: comparison between attribution method for hidden layers w.r.t. the predicted label. All plots are normalized per-layer by the largest attribution. Attention heatmap is obtained max pooling over heads and averaging across positions.

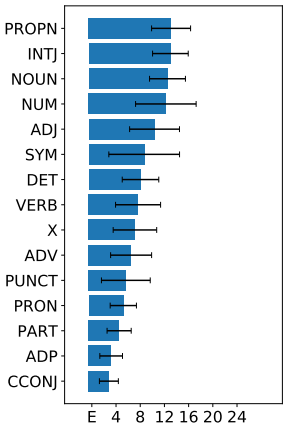


(a) Question inputs.

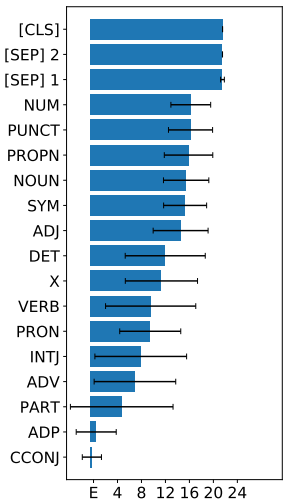
(b) Question hidden states.



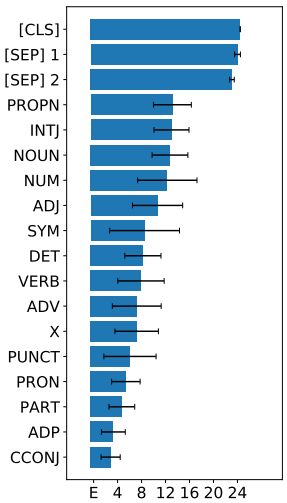
(c) Context inputs.



(d) Context hidden states.

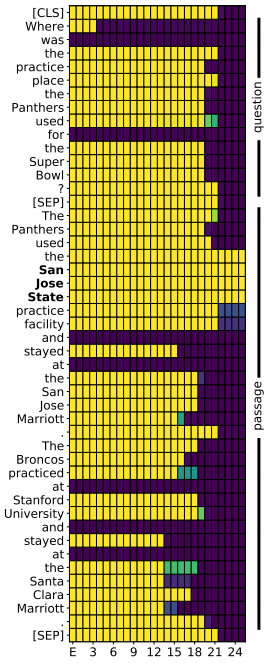


(e) All inputs.

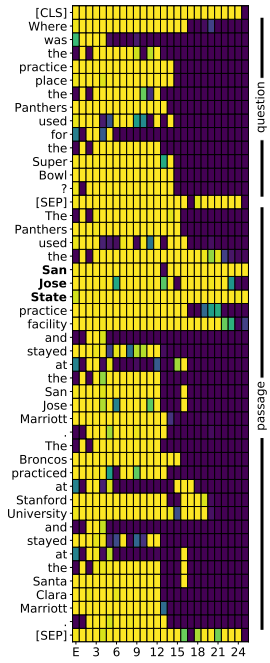


(f) All hidden states.

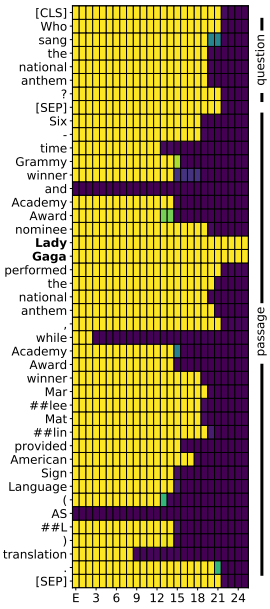
Figure 14: Question answering: average number of layers that predict to keep input tokens (a), (c) and (e) or hidden states (b), (d) and (f) aggregating by part-of-speech tag (POS) on validation set.



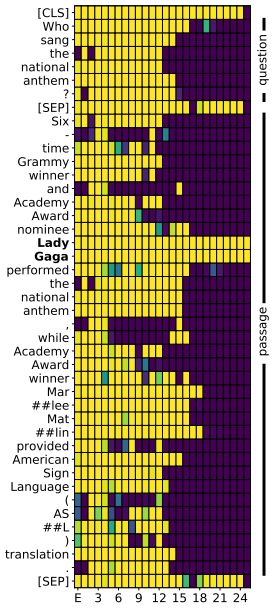
(a) Gating the input.



(b) Gating hidden states.



(c) Gating the input.



(d) Gating hidden states.

Figure 15: Expectation predicted by DIFFMASK to keep the inputs in (a) (c) and hidden states in (b) (d) on two different QA pairs. The correct answers is highlighted in bold.