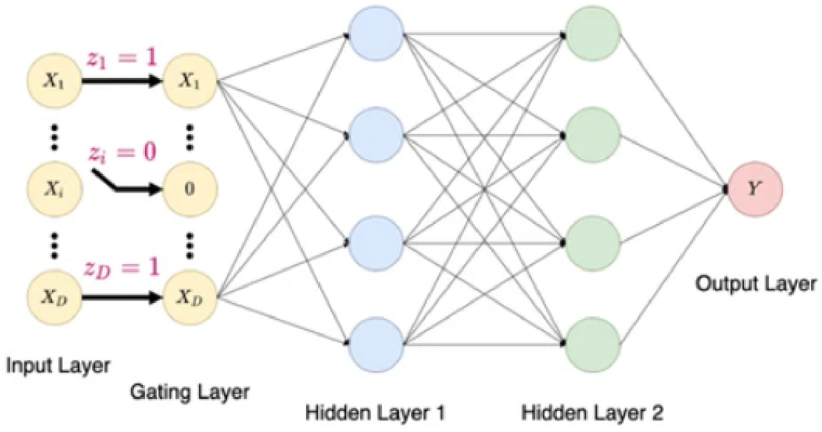
SToG blogpost

October 2025

$$z_i = \begin{cases} 1 & \text{w.p.} \quad \pi_i \\ 0 & \text{w.p.} \; 1 - \pi_i \end{cases}$$



This post introduces SToG — a compact Python library that brings learnable, stochastic feature selection into end-to-end deep learning. If you face high-dimensional inputs, want interpretable, sparse subsets of features, and still need the expressiveness of nonlinear models, SToG is for you. The library and a technical report with details, intuitions, and training tips are available in the project materials.

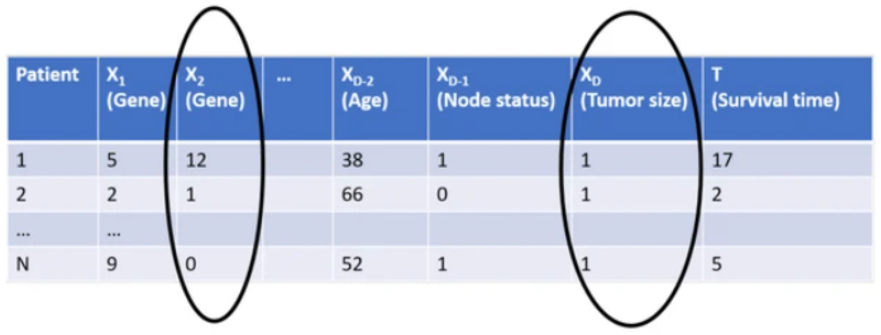
1 Why Feature Selection Matters

Feature selection is an essential step in preparing data for machine learning models. It involves choosing the most relevant features—or variables—that directly influence the output a model tries to predict. Why is this important?

Because real-world datasets often contain many features, but not all contribute meaningful information. Including irrelevant or redundant features can lead to complex models that take longer to train, risk overfitting (fitting noise rather than signal) and are harder to interpret

By selecting a smaller subset of key features, models become simpler, faster, and often more accurate. They are also easier to understand and explain, which is crucial in fields like healthcare or finance where decision transparency matters.

An example of dataset, where one feature is useless.



Patient	X_1 (Gene)	X_2 (Gene)	...	X_{D-2} (Age)	X_{D-1} (Node status)	X_D (Tumor size)	T (Survival time)
1	5	12		38	1	1	17
2	2	1		66	0	1	2
...	...						
N	9	0		52	1	1	5

2 Stochastic Gating for Feature Selection

Stochastic gating is a modern method designed to overcome some of these challenges. It works by associating a stochastic (random) "gate" variable with each feature. Each gate decides probabilistically whether the feature is "on" (used) or "off" (ignored) during model training.

Concretely, stochastic gates approximate selecting features by sampling from a relaxed Bernoulli distribution, where the probability of a gate being active is learned continuously. This lets the training process use gradient-based optimization to both learn the model and select features simultaneously.

SToG provides several complementary gating mechanisms you can use independently or mix, depending on your task.

Classical tools like LASSO use an ℓ_1 penalty and work well for *linear* models. But modern models are usually non-linear (deep neural networks) and we want sparse input also there. The idea of *stochastic gates* (STG) is to add a small stochastic layer in front of the network and train it together with the model to do feature selection.

2.1 From hard feature selection to stochastic gates

Assume we have input vectors $x \in R^D$ and a model $f_\theta(x)$ with parameters θ. For each feature we introduce a binary *gate* $s_d \in \{0, 1\}$. If $s_d = 0$, feature x_d is removed; if $s_d = 1$, we keep it. We can write the model with gates as

$$f_\theta(x \odot s),$$

where $\odot$ is element-wise product and $s = (s_1, \dots, s_D)$.

Let $L(\hat{y}, y)$ be some loss (for example, cross-entropy or squared error). The ideal feature selection problem is then

$$\min_{\theta,\, s} \; E_{X,Y}\, L\big(f_\theta(X \odot s), Y\big) \quad s.t. \quad \|s\|_0 \le k, \tag{1}$$

where $\|s\|_0$ is the number of non-zero entries in s (how many features we use).

This is a combinatorial problem: we would have to check many subsets of features. It is not realistic for high-dimensional data and cannot be optimized with gradient descent.

To fix this, the paper moves from *deterministic* gates s to *random* gates $\tilde{S}$. Each gate is a Bernoulli random variable:

$$\tilde{S}_d \sim Bernoulli(\pi_d), \qquad P(\tilde{S}_d = 1) = \pi_d.$$

We now optimize the expected loss:

$$\hat{R}(\theta, \pi) = E_{\hat{X},\hat{Y}} E_{\tilde{S}} \Big[L\big(f_\theta(\hat{X} \odot \tilde{S}), \hat{Y}\big) + \lambda \|\tilde{S}\|_0 \Big]. \tag{2}$$

Here $\lambda > 0$ controls how much we penalize the number of active features. Under this stochastic formulation

$$E\|\tilde{S}\|_0 = \sum_{d=1}^{D} P(\tilde{S}_d = 1) = \sum_{d=1}^{D} \pi_d,$$

so the regularizer becomes a simple sum of Bernoulli parameters.

The problem: $\tilde{S}$ is still discrete. We cannot backpropagate through samples of Bernoulli variables in a standard way. This is where *stochastic gates* come in.

2.2 Gaussian relaxation: the STG gate

Instead of sampling a Bernoulli variable, STG uses a continuous relaxation $z_d \in [0, 1]$. For each feature d we define a *gate parameter* μ_d and a Gaussian perturbation

$$\epsilon_d \sim \mathcal{N}(0, \sigma^2),$$

then compute

$$z_d = clip(\mu_d + \epsilon_d) = \max\big(0, \min(1,\, \mu_d + \epsilon_d)\big). \tag{3}$$

Vector $z = (z_1, \dots, z_D)$ is the vector of *stochastic gates.* Now the network receives input $x \odot z$ instead of x.

This simple formula has several nice properties:

- z_d is continuous, so we can compute gradients with respect to μ_d using the reparameterization trick.
- The gate is bounded in $[0, 1]$ thanks to clipping.

- The distribution is based on a Gaussian, which has lighter tails than logistic distributions used in other relaxations (like Concrete or Hard-Concrete).

The new training objective is

$$\min_{\theta,\,\mu} E_{\hat{X},\hat{Y}} E_Z \Big[L\big(f_\theta(\hat{X} \odot Z), \hat{Y}\big) + \lambda \|Z\|_0 \Big], \tag{4}$$

where Z is a random vector with coordinates z_d defined by eq:stg-gate.

Under this relaxation, the expected ℓ_0 term becomes

$$E\|Z\|_0 = \sum_{d=1}^{D} P(z_d > 0).$$

Because $z_d > 0$ exactly when $\mu_d + \epsilon_d > 0$, we get

$$P(z_d > 0) = P(\mu_d + \epsilon_d > 0) = P\Big(\epsilon_d > -\mu_d\Big) = \Phi\Big(\frac{\mu_d}{\sigma}\Big),$$

where Φ is the CDF of the standard normal distribution. So the regularizer becomes

$$\mathcal{R}(\mu) = \lambda \sum_{d=1}^{D} \Phi\Big(\frac{\mu_d}{\sigma}\Big). \tag{5}$$

This is a smooth, differentiable approximation of the number of selected features.

In practice we optimize eq:stg-objective using standard stochastic gradient descent: on each mini-batch we sample ϵ_d's, build z_d via eq:stg-gate, run forward and backprop, and update both θ and μ.

2.3 What happens after training

After the model is trained, we want a *deterministic* set of selected features. The simplest way is to remove the noise and use

$$\hat{z}_d = clip(\mu_d) = \max\big(0, \min(1,\, \mu_d)\big).$$

If the signal is strong, $\hat{z}_d$ usually goes close to 0 or 1, so we can interpret it directly: features with $\hat{z}_d \approx 1$ are selected, others are dropped. If needed, we can also add a threshold, for example $\hat{z}_d > 0.5$.

The nice side effect of using noise is the so-called "second chance" effect: even if a feature is temporarily suppressed, later noise samples can re-activate its gate and give it another chance to prove its usefulness during training.

3 StoG library

There will be description of implemented methods, library structure and a small demo.

4 Inspiring examples of application

There will be some examples of application this method and this library for different ML and DL tasks. The structure will be as follows: experimental setup and results, showing that STG is effective for feature selection task, And implementation of this method using our library is user-friendly. There will be: (a) a toy example where STG keeps only a few input coordinates, (b) one small table comparing accuracy and number of features for several methods.