SToG: Python Library for Feature Selection in Neural Networks

Altay Eynullayev Sergey Firsov Gleb Karpeev Denis Rubtsov

2025-12-01

Abstract / Motivation

Feature selection is a fundamental challenge in machine learning, particularly when dealing with high-dimensional data where the number of features can greatly exceed the number of samples. Traditional feature selection methods such as LASSO work well for linear models but struggle to capture nonlinear relationships inherent in complex datasets. Deep neural networks, while powerful, often suffer from overfitting and computational inefficiency when processing datasets with thousands or millions of features.

SToG (Stochastic Gates for Feature Selection) is a Python library that addresses these challenges by implementing four state-of-the-art deep learning-based feature selection methods. The library enables end-to-end differentiable feature selection through stochastic gating mechanisms, allowing neural networks to simultaneously learn predictive models while identifying the most relevant features. This approach is particularly valuable in domains such as genomics, medical diagnostics, finance, and any field where interpretability and computational efficiency are critical.

The motivation behind SToG is threefold: (1) to provide researchers and practitioners with robust tools for nonlinear feature selection, (2) to enable feature selection with limited labeled data through self-supervised learning, and (3) to support both static feature pruning and dynamic conditional computation for improved inference efficiency.

Plan and Structure

This technical report is organized as follows:

1. Introduction

- Background on feature selection challenges
- Overview of the SToG library and its capabilities
- Key contributions and target applications

2. Implemented Methods

Each of the four feature selection methods is described in detail:

2.1 STG (Stochastic Gates)

2.2 SEFS (Self-Supervision Enhanced Feature Selection)

2.3 Gumbel Softmax Channel Selection

2.4 Gradient Estimators for Stochastic Neurons

3. Library Architecture and Usage

- Installation instructions
- API design and key classes
- Code examples for each method
- Integration with PyTorch and other frameworks

4. Experimental Validation

- Benchmark datasets (synthetic and real-world)
- Performance metrics (accuracy, feature selection quality, computational cost)
- Comparative analysis with baseline methods

5. Implementation Details

- Network architectures
- Hyperparameter settings and tuning strategies
- Training procedures and optimization

6. Discussion and Future Work

- Strengths and limitations of each method
- Recommended use cases
- Potential extensions and research directions

7. Conclusion

- Summary of key findings
- Practical recommendations for users

---

1. Introduction

1.1 Background and Motivation

Feature selection aims to identify a small subset of relevant features from a potentially very large feature space. This task is crucial for several reasons:

- **Reduced computational cost**: Fewer features mean faster training and inference
- **Improved generalization**: Removing irrelevant features reduces overfitting
- **Enhanced interpretability**: Simpler models are easier to understand and explain
- **Lower experimental costs**: In domains like genomics, measuring fewer biomarkers reduces expenses

Traditional approaches to feature selection fall into three categories:

1. **Filter methods**: Select features based on statistical measures (e.g., correlation, mutual information) independent of the learning algorithm
2. **Wrapper methods**: Evaluate feature subsets by training a model and measuring performance (e.g., recursive feature elimination)
3. **Embedded methods**: Perform feature selection as part of the model training process (e.g., LASSO, Elastic Net)

While LASSO and related methods excel at linear feature selection, they are limited to linear models and suffer from weight shrinkage. Modern deep learning applications require feature selection methods that can:

- Handle **nonlinear relationships** between features and targets
- Work effectively with **high-dimensional, low-sample-size** datasets
- Provide **end-to-end differentiable** training pipelines
- Account for **feature correlations** and redundancy
- Enable **conditional computation** for efficient inference

SToG addresses these requirements by implementing four complementary approaches based on stochastic gating mechanisms.

1.2 SToG Library Overview

SToG is an open-source Python library that provides implementations of four cutting-edge feature selection methods:

1. **STG (Stochastic Gates)**: Uses Gaussian-based continuous relaxation of Bernoulli distributions for stable gradient estimation
2. **STE (State-Through Estimator)**: Binary gates with gradient approximation
3. **Gumbel Softmax**: Categorical relaxation for feature gating

4. **Correlated STG**: Extension for handling correlated features

Each method can be used independently or in combination, depending on the specific requirements of the task. Conceptual diagram showing feature gating mechanism.

Conceptual diagram showing feature gating mechanism

Figure 1: Conceptual diagram showing feature gating mechanism

Figure 1: Conceptual overview of stochastic gating for feature selection. Each input feature x_d is multiplied by a learnable stochastic gate $z_d \in \{0, 1\}$, which determines whether the feature is selected ($z_d = 1$) or discarded ($z_d = 0$).

---

2. Implemented Methods

This section provides detailed descriptions of the four feature selection methods implemented in SToG, including their theoretical foundations, algorithms, and practical considerations.

2.1 STG: Stochastic Gates with Gaussian Relaxation

2.1.1 Motivation and Core Idea The STG method, proposed by Yamada et al. (ICML 2020), addresses the challenge of optimizing the 0 norm—the count of non-zero features—which is combinatorially hard and non-differentiable. Traditional 1 regularization (LASSO) provides a convex approximation but is restricted to linear models and causes unwanted weight shrinkage.

STG introduces a **probabilistic approach** by modeling feature selection as sampling from independent Bernoulli distributions. Each feature x_d has an associated Bernoulli gate with probability μ_d, which determines whether the feature is active. By using a continuous relaxation, STG makes this process fully differentiable, enabling gradient-based optimization.

2.1.2 Mathematical Formulation Let $\mathbf{x} \in \mathbb{R}^D$ be the input features and y be the target variable. The goal is to learn a function $f_\theta(\mathbf{x} \odot \mathbf{z})$ that predicts y while using only a small subset of features, where $\mathbf{z} \in \{0, 1\}^D$ is a binary vector indicating which features are selected.

Stochastic Gate Definition: For each feature dimension $d = 1, \ldots, D$, the stochastic gate is defined as:

$$z_d = \max(0, \min(1, \mu_d + \epsilon_d))$$

where: - μ_d is a learnable parameter controlling the gate probability - $\epsilon_d \sim \mathcal{N}(0, \sigma^2)$ is Gaussian noise with fixed variance σ^2 - The max-min operation creates a "hard sigmoid" that clips values to [0,1]

This formulation approximates sampling from a Bernoulli distribution with probability $p_d = \Phi(\mu_d/\sigma)$, where Φ is the standard Gaussian CDF.

Objective Function: The training objective combines a task-specific loss with a regularization term encouraging sparsity:

$$\mathcal{L}(\theta, \mu) = \mathbb{E}_{\mathbf{z} \sim p_\mu} [L(f_\theta(\mathbf{x} \odot \mathbf{z}), y)] + \lambda \sum_{d=1}^{D} \Phi\left(\frac{\mu_d}{\sigma}\right)$$

where: - $L(\cdot)$ is the task loss (cross-entropy for classification, MSE for regression) - λ is the regularization strength hyperparameter - $\Phi(\mu_d/\sigma)$ approximates the expected value of z_d, representing the probability that feature d is active

---

2.2 SEFS: Self-Supervision Enhanced Feature Selection

2.2.1 Motivation A major challenge in real-world applications is the scarcity of labeled data, especially in domains like medicine and genomics where labeling is expensive or requires expert knowledge. While many unlabeled samples may be available, traditional supervised feature selection methods cannot leverage this data.

SEFS (Self-Supervision Enhanced Feature Selection), proposed by Lee et al. (ICLR 2022), addresses this by introducing a **two-phase training procedure**: 1. **Self-Supervision Phase**: Pre-train an encoder on unlabeled data using pretext tasks 2. **Supervision Phase**: Fine-tune for feature selection using limited labeled data

Additionally, SEFS introduces **correlated gating** via Gaussian copula to handle multicollinear features—a common issue in high-dimensional datasets where features are highly correlated.

2.2.2 Two-Phase Training Phase 1: Self-Supervision (Unlabeled Data)

The goal is to pre-train an encoder f_ϕ that learns informative representations from partial feature sets. Two pretext tasks are jointly optimized:

1. **Feature Vector Reconstruction**: Given a masked input $\mathbf{x} \odot \mathbf{m}$, reconstruct the original $\mathbf{x}$
2. **Gate Vector Estimation**: Predict which features were masked (estimate $\mathbf{m}$)

The architecture consists of: - **Encoder** f_ϕ: Maps input to latent representation $\mathbf{z} = f_\phi(\mathbf{x} \odot \mathbf{m})$ - **Feature Vector Estimator** h_1: Reconstructs $\hat{\mathbf{x}} = h_1(\mathbf{z})$ - **Gate Vector Estimator** h_2: Predicts $\hat{\mathbf{m}} = h_2(\mathbf{z})$

The self-supervision loss is:

$$\mathcal{L}_{\text{SS}} = \mathbb{E}_{\mathbf{x},\mathbf{m}} \left[\|\mathbf{x} - \hat{\mathbf{x}}\|_2^2 + \alpha \sum_d \left(m_d \log \hat{m}_d + (1 - m_d) \log(1 - \hat{m}_d) \right) \right]$$

where α balances the two objectives.

Phase 2: Supervision (Labeled Data)

After pre-training, the encoder is fine-tuned for feature selection: - The estimators h_1, h_2 are discarded - A new predictor f_ψ is added: $\hat{y} = f_\psi(f_\phi(\mathbf{x} \odot \mathbf{m}))$ - Both ϕ and selection probabilities π are optimized

The supervision objective is:

$$\mathcal{L}_{\text{Sup}} = \mathbb{E}_{\mathbf{x},y,\mathbf{m}\sim p_\pi} \left[L(\hat{y}, y) + \lambda \sum_d \pi_d \right]$$

2.2.3 Correlated Gates via Gaussian Copula Real-world features often exhibit strong correlations (e.g., gene expressions, financial indicators). Independent gating mechanisms can select redundant features, while correlated gating encourages competition among related features.

SEFS models the gate vector $\mathbf{m}$ as a **multivariate Bernoulli** distribution whose correlation structure is derived from the input features:

1. **Correlation Matrix**: Compute $\mathbf{R}$ from the training data:

$$R_{ij} = \text{Corr}(X_i, X_j)$$

2. **Gaussian Copula**: Generate correlated uniform variables $\mathbf{u} = (u_1, \ldots, u_D)$ via:

$$\mathbf{u} = \Phi_{\mathbf{R}}(\mathbf{v})$$

   where $\mathbf{v} \sim \mathcal{N}(\mathbf{0}, \mathbf{R})$ and $\Phi_{\mathbf{R}}$ is the Gaussian copula CDF.

3. **Multivariate Bernoulli Sampling**: Sample gates as:

$$m_d = \mathbb{1}_{u_d < \pi_d}$$

Reparameterization: For differentiability, use a continuous relaxation:

$$m_d = \sigma\left(\frac{1}{\tau} \left(\log \pi_d - \log(1 - \pi_d) + \log u_d - \log(1 - u_d) \right) \right)$$

where $\sigma(\cdot)$ is the sigmoid function and τ is a temperature parameter.

2.2.4 Benefits of Correlated Gating Correlated gating provides two key advantages:

1. **During Self-Supervision**: Prevents trivial solutions where the network memorizes correlations between features. If features X_i and X_j are highly correlated and both are masked together, the reconstruction task becomes more challenging and forces the encoder to learn robust representations.
2. **During Supervision**: Encourages selection of only the most informative feature from a correlated group. Highly correlated features compete against each other, reducing redundancy in the selected subset.

Visualization of correlated vs independent gating

Figure 2: Visualization of correlated vs independent gating

Figure 3: Correlated gating (left) vs. independent gating (right). With correlation, related features are more likely to be masked together, encouraging the model to learn non-redundant representations.

2.3 Gumbel Softmax for Channel Selection

2.3.1 Context: Neural Network Pruning and Conditional Computation While STG and SEFS focus on feature selection at the input layer, the Gumbel Softmax method by Herrmann et al. addresses **channel selection** within neural network layers. This approach has two main applications:

1. **Network Pruning**: Permanently remove unnecessary channels to create a smaller, faster model
2. **Conditional Computation**: Dynamically decide which channels to compute based on the input, enabling adaptive inference

Both applications aim to reduce computational cost (measured in FLOPs) while maintaining accuracy.

2.3.2 Gumbel-Softmax Reparameterization The challenge in channel selection is similar to feature selection: making discrete decisions (keep or remove a channel) in a differentiable way. The **Gumbel-Softmax trick** provides an elegant solution.

Standard Gumbel-Max Trick (non-differentiable): To sample from a categorical distribution with probabilities $\mathbf{p} = (p_1, \ldots, p_K)$:

$$z = \arg\max_k \left(\log p_k + G_k\right)$$

where $G_k \sim \text{Gumbel}(0, 1)$ are i.i.d. Gumbel random variables.

Gumbel-Softmax Relaxation (differentiable): Replace the $\arg\max$ with a softmax:

$$z_k = \frac{\exp\left((\log p_k + G_k)/\tau\right)}{\sum_{j=1}^{K} \exp\left((\log p_j + G_j)/\tau\right)}$$

where $\tau > 0$ is a temperature parameter. As $\tau \to 0$, the softmax approaches a one-hot vector.

For Binary Gates (keep/remove a channel):

$$z = \sigma\left(\frac{1}{\tau}\left(\log\frac{p}{1-p} + G\right)\right)$$

where p is the probability of keeping the channel, $G = G_1 - G_2$ with $G_1, G_2 \sim \text{Gumbel}(0, 1)$, and $\sigma(\cdot)$ is the sigmoid function.

2.3.3 Batch Activation Loss To control the sparsity of the network, the Gumbel Softmax method introduces a novel **batch activation loss**:

$$\mathcal{L}_B = \left(\frac{1}{BG}\sum_{i=1}^{G}\sum_{j=1}^{B} z_{i,j} - t\right)^2$$

where: - B is the batch size - G is the number of gates (channels) - $z_{i,j}$ is the gate value for gate i and sample j - $t \in [0, 1]$ is the target activation rate (e.g., $t = 0.5$ means 50% of channels should be active)

This loss can be decomposed into two terms:

$$\mathcal{L}_B = \underbrace{(\mathbb{E}[\bar{z}] - t)^2}_{\text{Bias term}} + \underbrace{\text{Var}[\bar{z}]}_{\text{Variance term}}$$

where $\bar{z} = \frac{1}{BG} \sum_{i,j} z_{i,j}$ is the mean activation.

Key Properties: 1. The bias term encourages the overall activation rate to match the target t 2. The variance term encourages **polarization**: gates should be close to 0 or 1, not hovering around intermediate values 3. Unlike previous methods, this allows flexibility—each gate can learn its own activation rate, as long as the global average matches t

---

2.4 Gradient Estimators for Stochastic Neurons

2.4.1 Straight-Through Estimator (Biased) The **straight-through (ST) estimator** is a simple heuristic that treats the hard threshold as if it were an identity function during backpropagation:

$$\text{Forward pass: } h = \mathbb{1}_{a>0}$$
$$\text{Backward pass: } \frac{\partial L}{\partial a} = \frac{\partial L}{\partial h}$$

In other words, gradients are copied directly through the threshold without modification. This is clearly biased (the true gradient is zero), but it is simple and often works well in practice.

Variant: Multiply by sigmoid derivative:

$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial h} \cdot \sigma(a) \cdot (1 - \sigma(a))$$

Empirically, the simple version (without multiplying by $\sigma'(a)$) often performs better.

Advantages: - Extremely simple to implement - Low variance - Computationally efficient

Disadvantages: - Biased estimator with no theoretical guarantees - May not converge to the correct solution in all cases

---

3. Library Architecture and Usage

We demonstrate the usage and performance of realised methods on several selection tasks. First, we are going to compare methods on the task of feature selection, when there is only one selection layer at the beginning of neural network.

3.1 Feature selection task

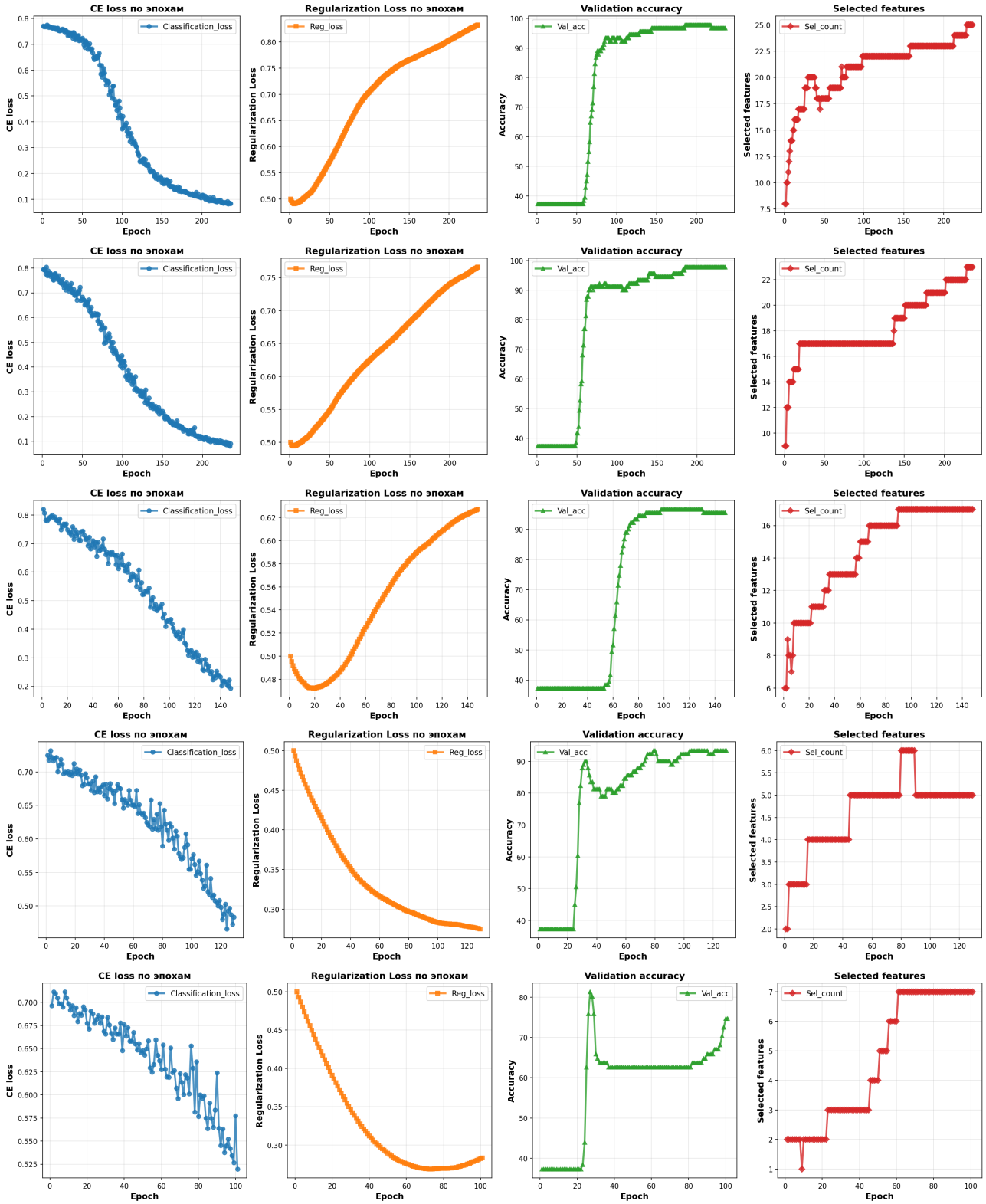

Figure 3: Feature selection using STG method on the breast cancer dataset with *a)* $\lambda = 0.0001$, *b)* $\lambda = 0.001$, *c)* $\lambda = 0.01$, *d)* $\lambda = 0.05$, *e)* $\lambda = 0.1$

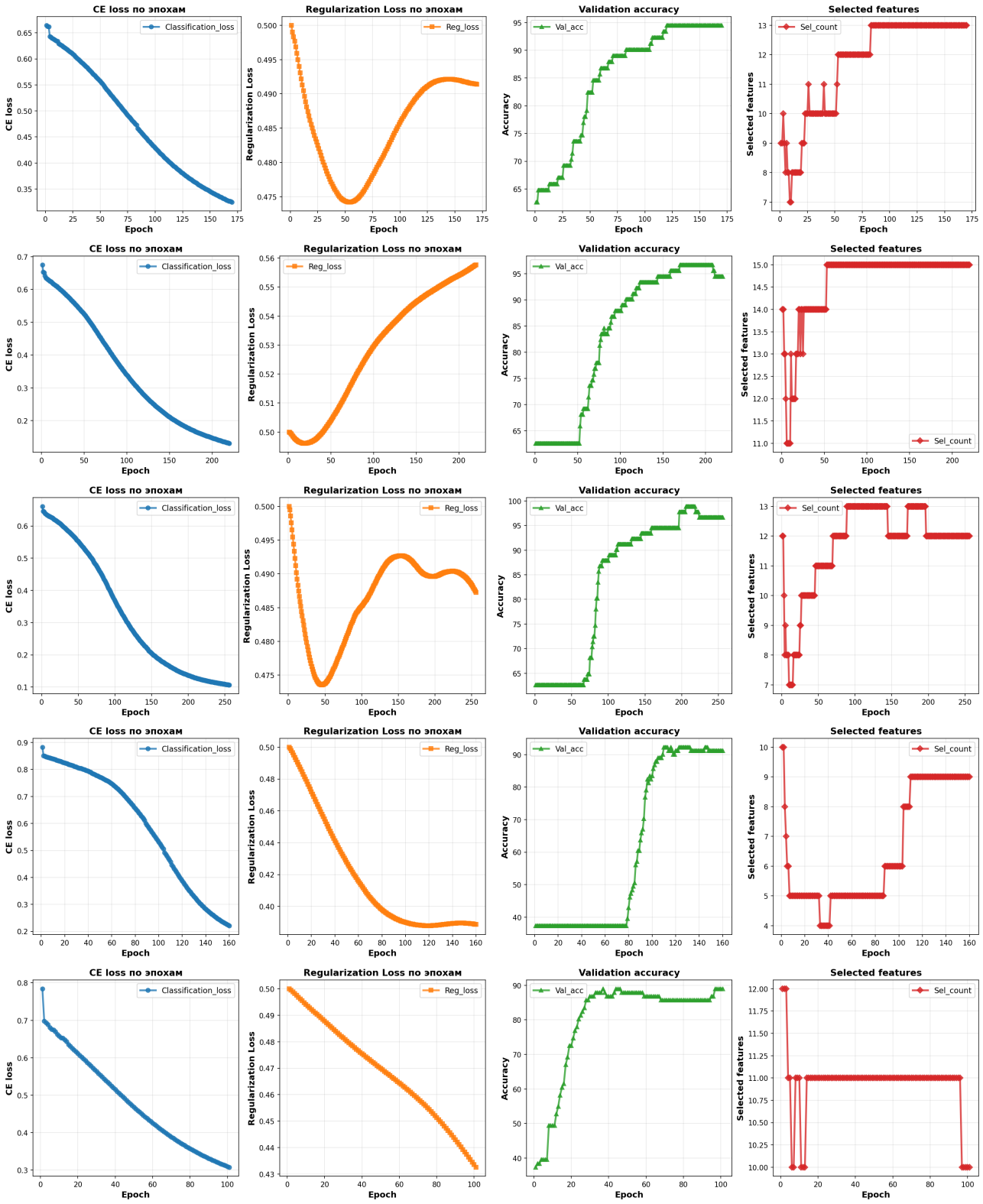

Figure 4: Feature selection using STE method on the breast cancer dataset with *a)* $\lambda = 0.0001$, *b)* $\lambda = 0.001$, *c)* $\lambda = 0.01$, *d)* $\lambda = 0.05$, *e)* $\lambda = 0.1$

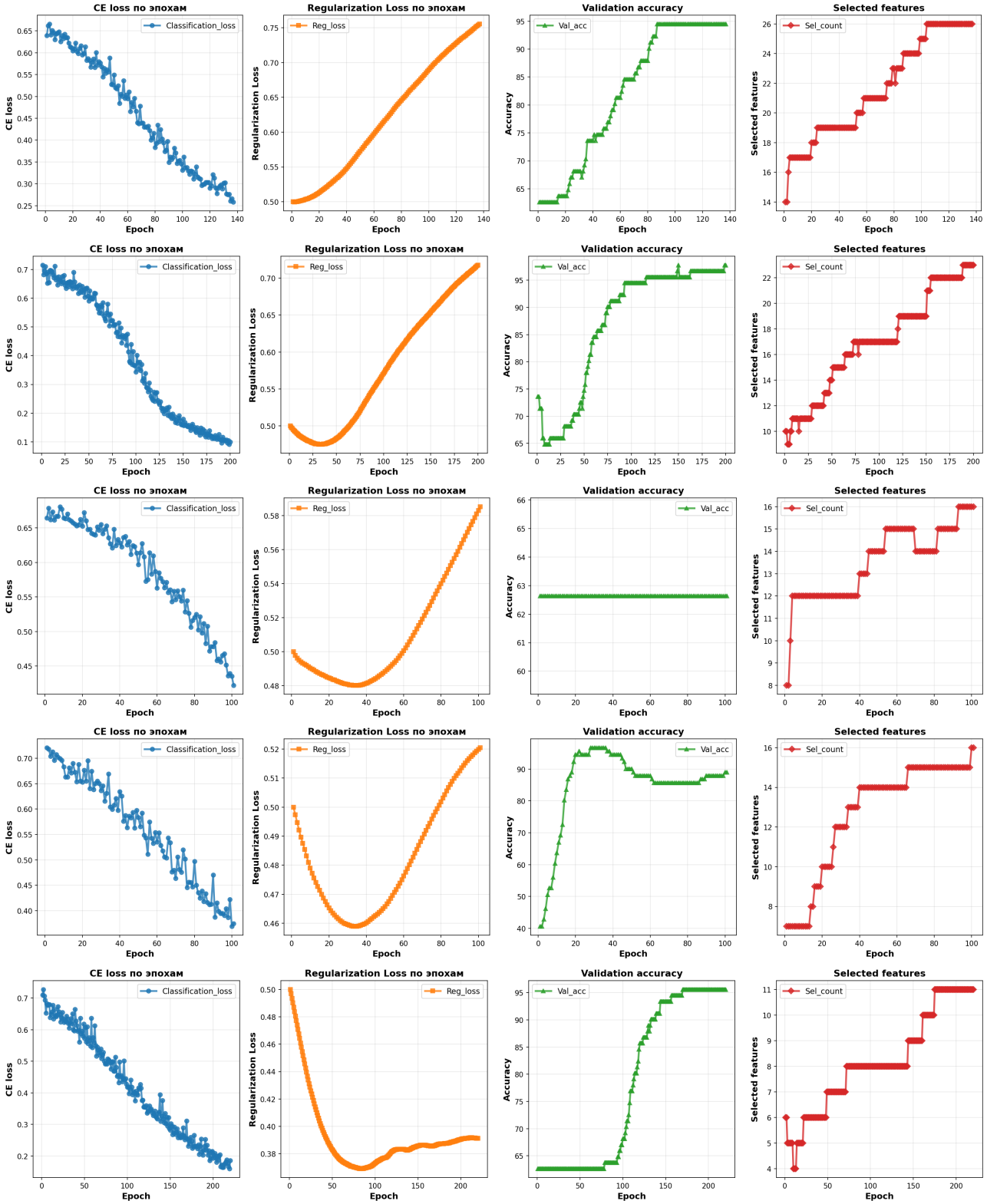

Figure 5: Feature selection using Gumbel method on the breast cancer dataset with *a)* $\lambda = 0.0001$, *b)* $\lambda = 0.001$, *c)* $\lambda = 0.01$, *d)* $\lambda = 0.05$, *e)* $\lambda = 0.1$