Calculating bounce actions with Neural Networks and extensions to the code EVADE

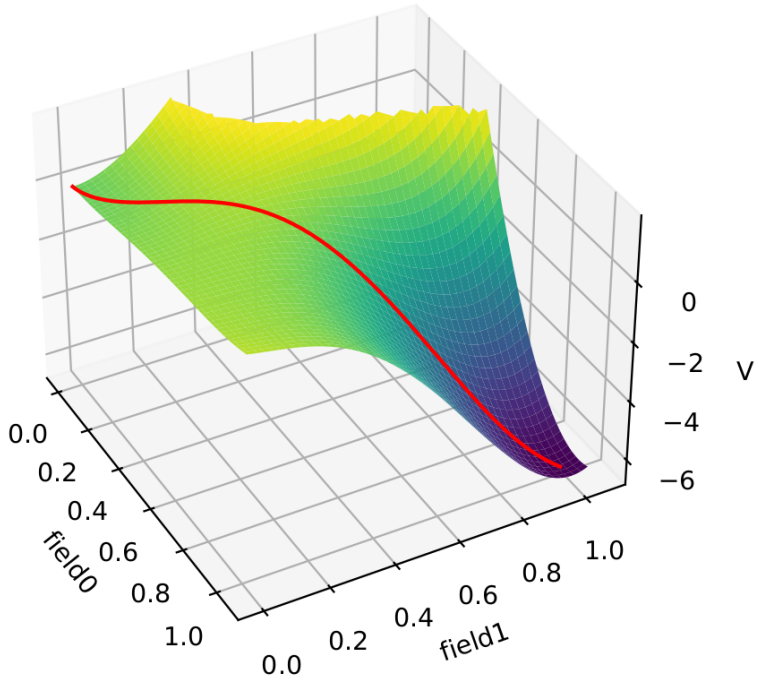
Fabio Campello

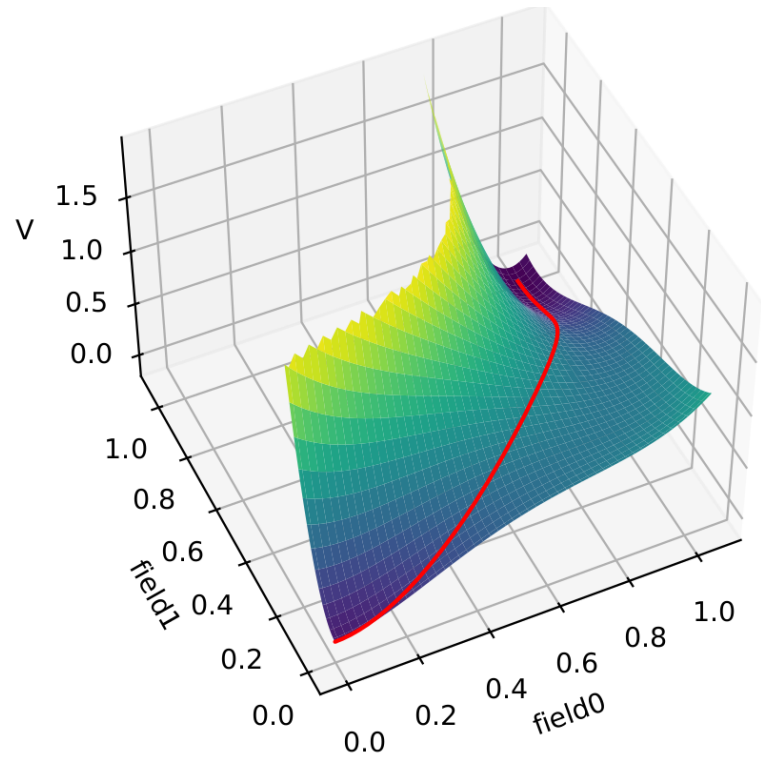
29.02.2024

Content

False Vacuum Decay

- System in a metastable state
- Quantum tunneling to a deeper minimum, $\Gamma \propto e^{-B}$
- Dominant contribution from bounce action B
- Bounce solution $\phi_B(\rho)$ solves $\frac{d^2\phi}{d\rho^2} + \frac{3}{\phi}\frac{d\phi}{d\rho} - \nabla V(\phi) = 0$ with $\phi_B(\infty) = \phi_{\text{false}}$ and $\left.\frac{d\phi}{d\rho}\right|_{\rho=0} = 0$
- Bounce action is obtained from the bounce solution

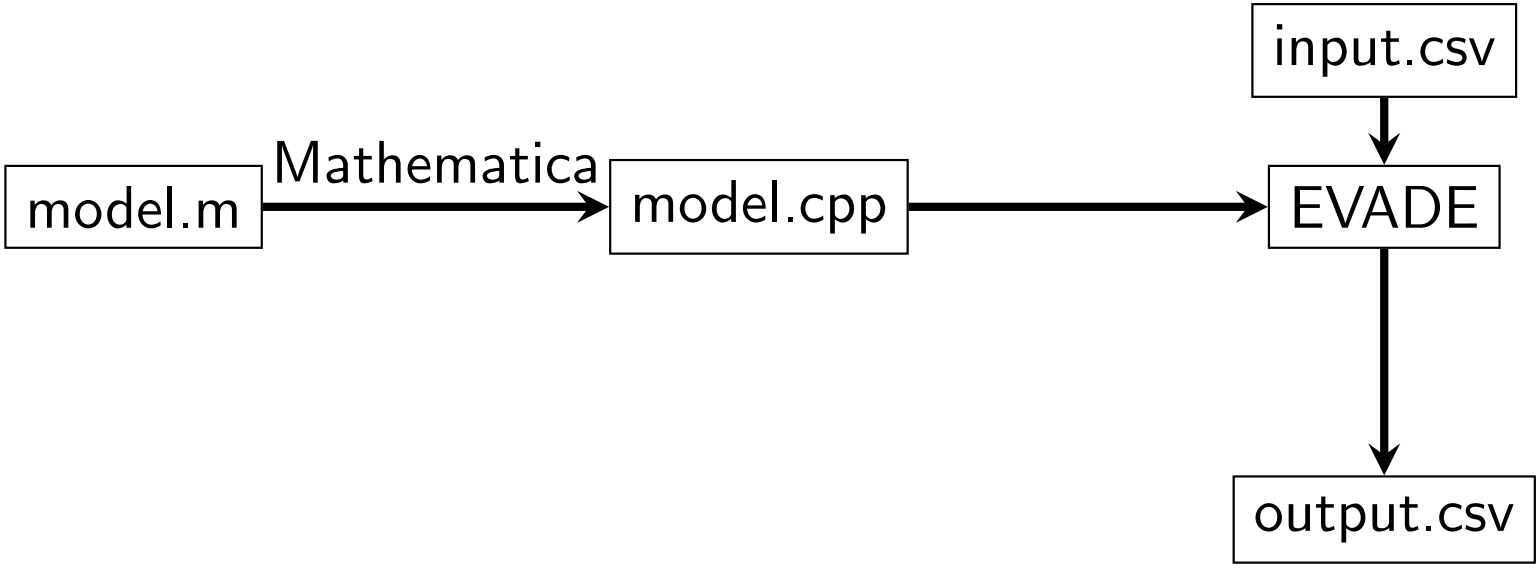




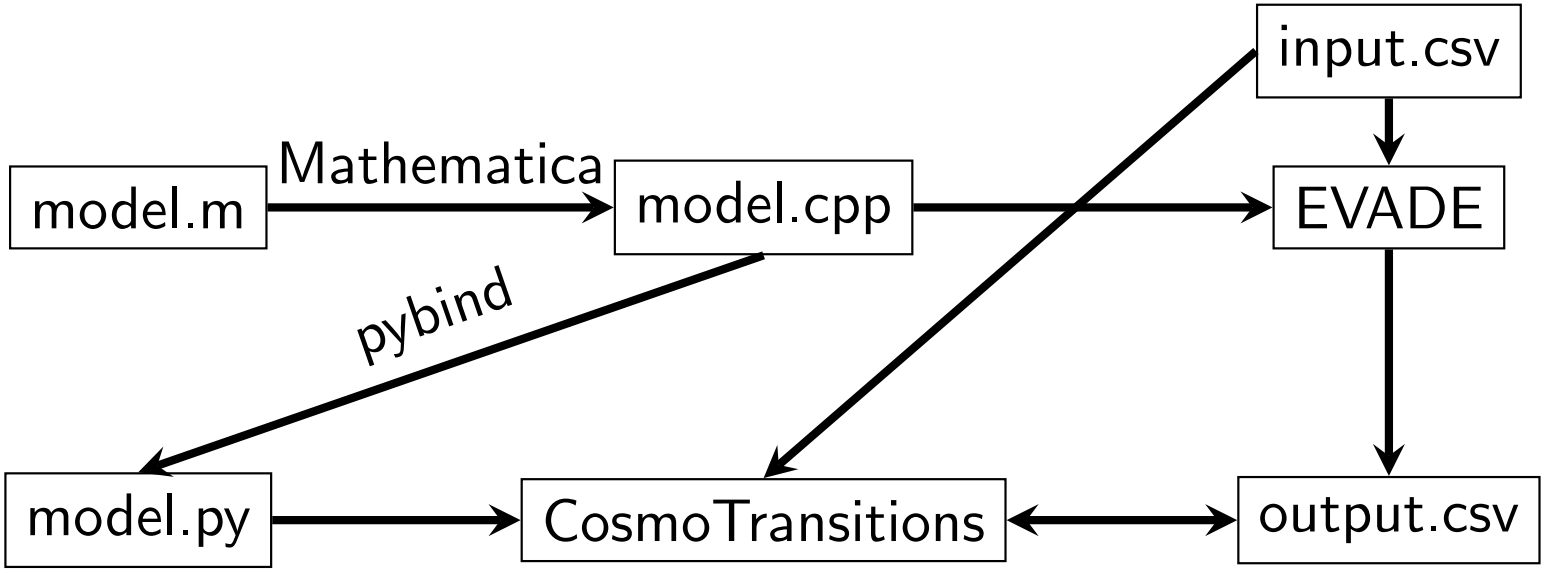
EVADE

- Any quartic potential of n fields
- Find all *extrema* using PHC, up to 3^n
- Straight path approximation, $V(\varphi) = \lambda\varphi^4 - A\varphi^3 + m^2\varphi^2$
- Use semianalytic result, $B = B(\lambda, A, m)$
- Compare to the age of the universe

Adding path deformation to EVADE



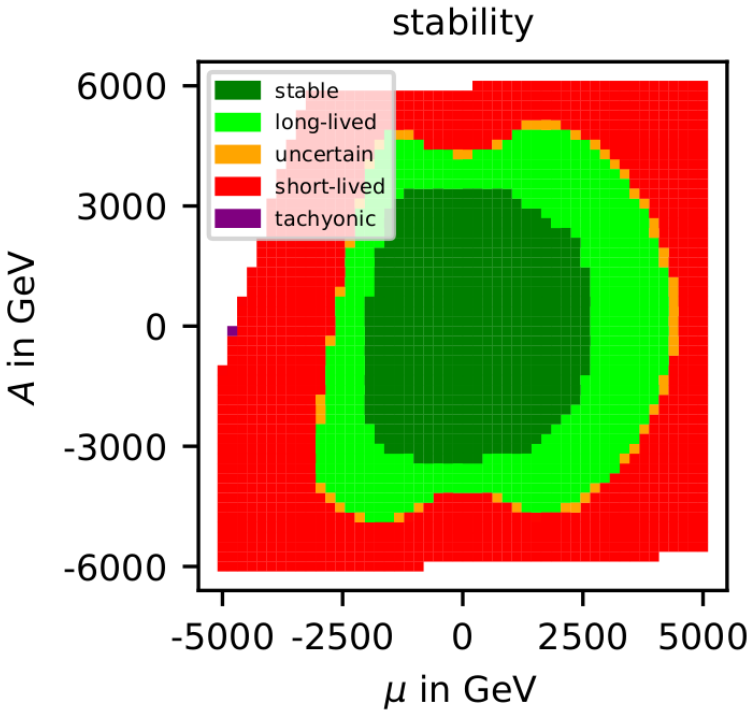
Adding path deformation to EVADE



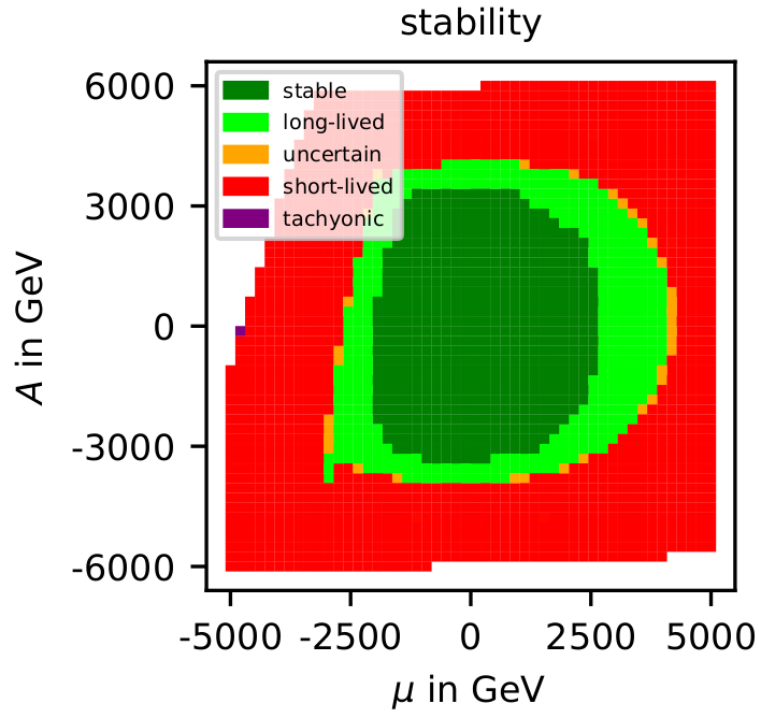
- Minimal additional setup
- Combines efficiency of EVADE with increased accuracy of using path deformation

EVADE path deformation results

EVADE



EVADE + Cosmo



- Path deformation can shift the border of short-lived regions
- Additional parameter space excluded
- Computing path deformation for this plot (2300 MSSM points) took around 90s using 12 CPU cores

Neural network approach - Definitions

- Treat neural network as a function
- Neural networks can approximate *any* function

The neural net will model the tunneling path $\phi(\rho)$
Choose a ρ_{max} and discretize the path into n_ρ steps
Define the loss: $\mathcal{L} = \mathcal{L}_{\text{eq}} + n_\rho \mathcal{L}_{\text{b}}$

$$\mathcal{L}_{\text{eq}} = \sum_i \left(\frac{d^2\phi(\rho_i)}{d\rho^2} + \frac{3}{\phi(\rho_i)} \frac{d\phi(\rho_i)}{d\rho} - \nabla V(\phi(\rho_i)) \right)^2$$

$$\mathcal{L}_{\text{b}} = \left(\frac{d\phi(\rho_0)}{d\rho} \right)^2 + (\phi(\rho_{\text{max}}) - \phi_{\text{false}})^2$$

Neural network prefers values between 0 and 1
$\Rightarrow \phi(\rho) = \phi_{\text{true}} + \text{NN}(\rho)(\phi_{\text{false}} - \phi_{\text{true}})$

Neural network approach - Training process

$$\mathcal{L}_{\text{eq}} = \sum_i \left(\frac{d^2\phi(\rho_i)}{d\rho^2} + \frac{3}{\phi(\rho_i)} \frac{d\phi(\rho_i)}{d\rho} - \nabla V(\phi(\rho_i)) \right)^2$$

$$\mathcal{L}_{\text{b}} = \left(\frac{d\phi(\rho_0)}{d\rho} \right)^2 + (\phi(\rho_{\text{max}}) - \phi_{\text{false}})^2$$

- Equation has a trivial solution $\phi(\rho) = \phi_{\text{false}}$
- Random initialization $\rightarrow$ Network finds trivial solution
- Solution: Two step training process

Randomly initialize the network, then train a few epochs on $\mathcal{L}_{\text{init}} = \sum_i (NN(\rho_i) - \frac{\rho_i}{\rho_{\text{max}}})^2$

Continue training with the correct Loss function

Neural network approach - Training process

- No inference $\rightarrow$ No overfitting
- Single batch training $\rightarrow$ almost no randomness
- Danger of getting stuck in local minima
- Danger of falling back into trivial solution

$\Rightarrow$ Choosing learning rate correctly is crucial

Neural network approach - Implementation overview

- Option 1: Use only existing tensorflow operations via python
- Option 2: Write custom tensorflow operation in C++

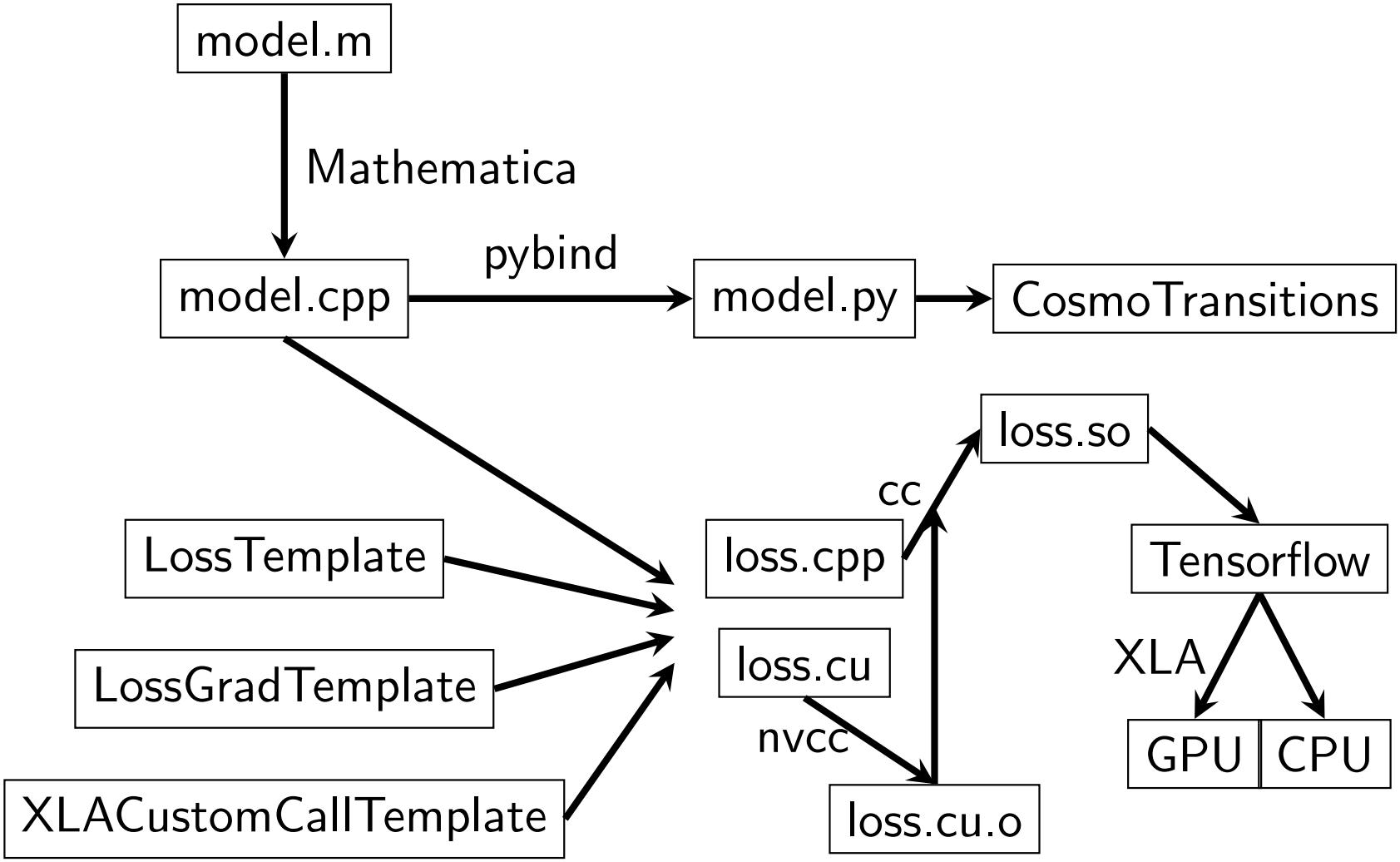
Python option:

- Easier to implement, no additional code to compile
- Use tensorflows automatic differentiation
- C++ model files can not be used
- Graph creation for complicated models is very slow

C++ option:

- C++ for Loss and its gradient, separate for CPU and GPU
- Need to use finite differences
- Use already available C++ model files
- Overall better performance

Neural network approach - Implementation overview



Neural network approach - Measure for method comparison

Given $\phi_{\text{cosmo}}(\rho)$ and $\phi_{\text{NN}}(\rho)$, which is more accurate?

- $B[\phi] = 2\pi^2 \int_0^\infty d\rho \rho^3 \left[\frac{1}{2} \left(\frac{d\phi(\rho)}{d\rho} \right)^2 + V(\phi(\rho)) \right] = I_K[\phi] + I_V[\phi]$
- One can show that $I_K[\phi_B] = -2I_V[\phi_B]$
- $\Rightarrow B[\phi_B] = -I_V[\phi_B] = \frac{1}{2} I_K[\phi_B]$

For a solution all three methods of calculating the Bounce action must give the same result.

Neural network approach - Hyperparameter

Discretization

- $\rho_{\max}$ set to equal CosmoTransitions
- n_ρ: 1000

NN parameters

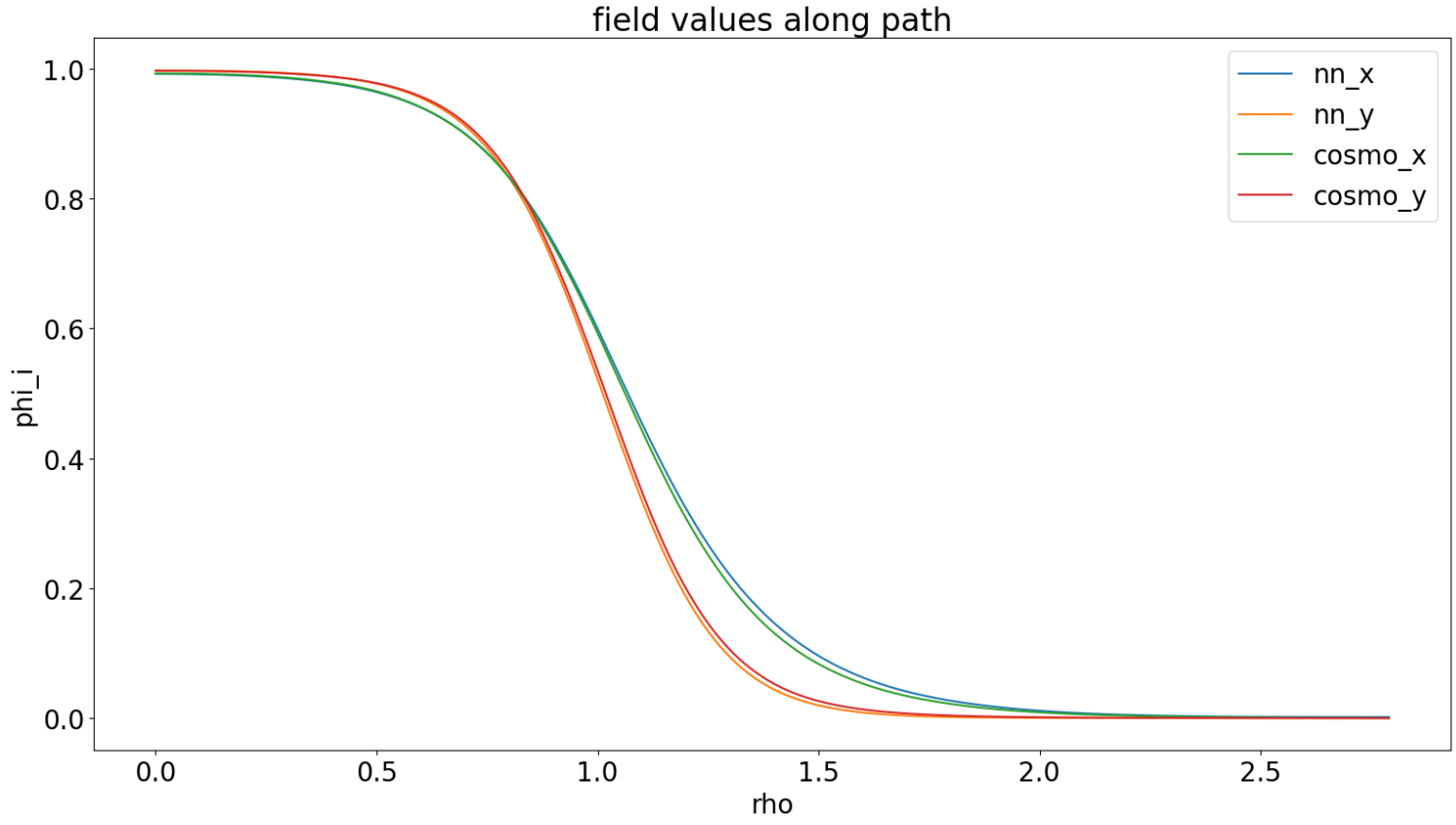
- Hidden layers: 5
- Neurons in each layer: 10
- Activation: tanh

Training parameters

- Optimizer: Adam
- Initialization epochs: 1000
- Initialization learning rate: 10^{-2}
- Training epochs: 20000
- Training learning rate: 10^{-3}

Neural network approach - Results

$$V(x,y) = (x^2 + 5y^2)\left(5(x-1)^2 + (y-1)^2\right) + 80(y^4 - y^3)$$

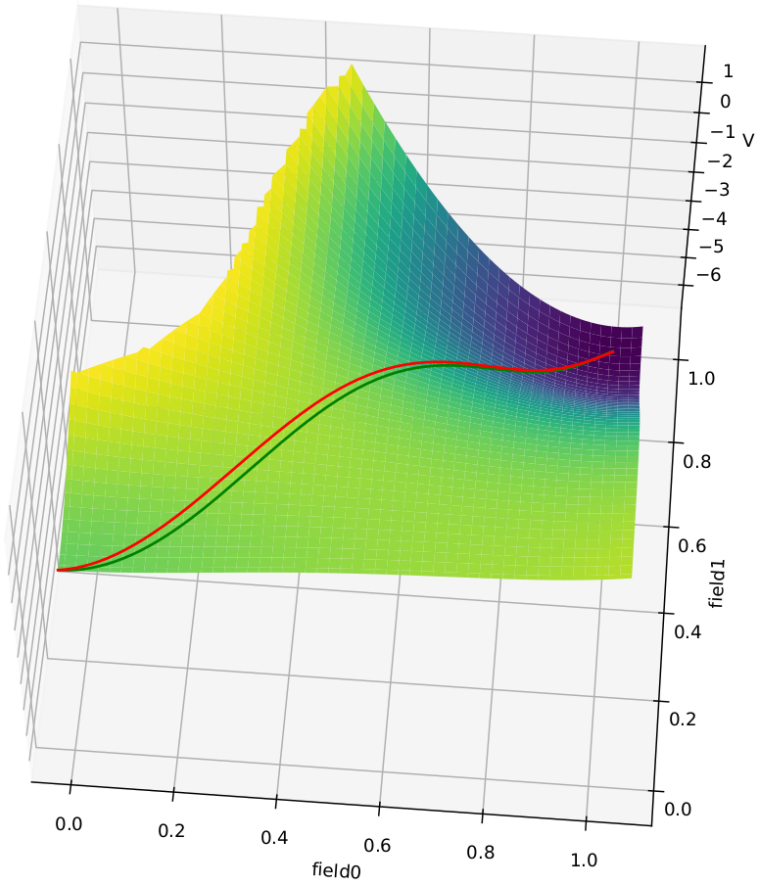


Cosmo: [13.329, 13.278, 13.228], NN: [13.168, 13.167, 13.165]

Runtime: Cosmo single thread: 0.6s, NN on RTX 3060 Ti: 12s

Neural network approach - Results

$$V(x,y) = (x^2 + 5y^2)\left(5(x-1)^2 + (y-1)^2\right) + 80(y^4 - y^3)$$

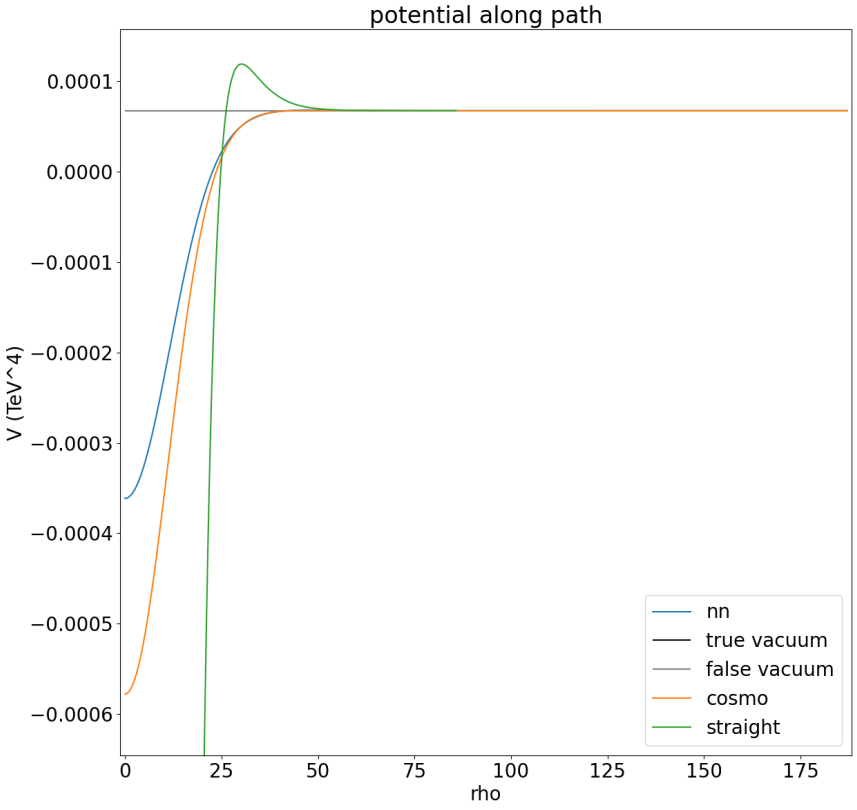


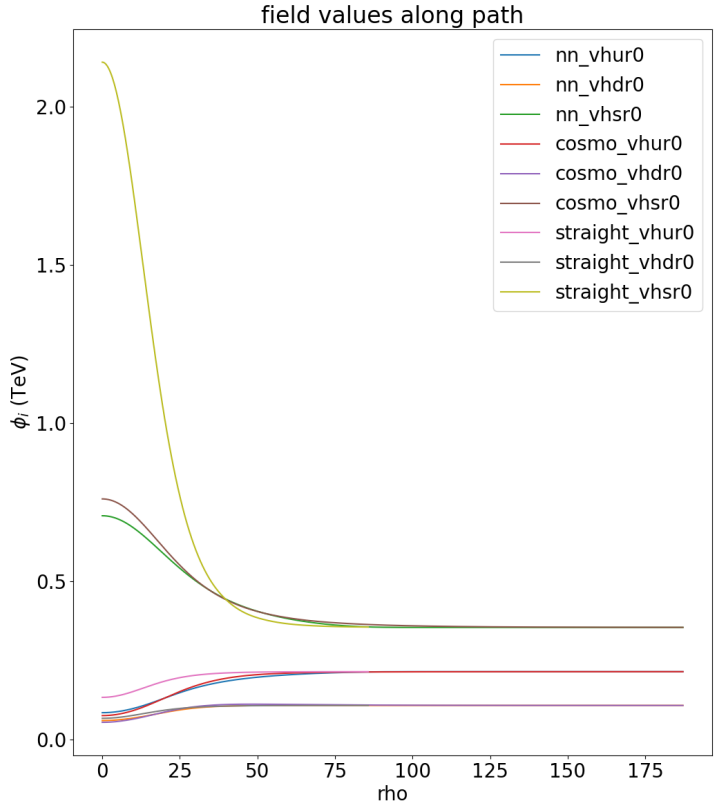
Cosmo: [13.329, 13.278, 13.228], NN: [13.168, 13.167, 13.165]
Runtime: Cosmo single thread: 0.6s, NN on RTX 3060 Ti: 12s

Neural network approach - Results

NMSSM potential, 3 fields non-zero

Learning rate was set to 10^{-2} and epochs to 30000





Cosmo: [351.57, 351.95, 352.32], NN: [403.80, 366.06, 268.33]

Straight: 2910

Runtime: Cosmo single thread: 0.8s, NN on RTX 3060 Ti: 29s

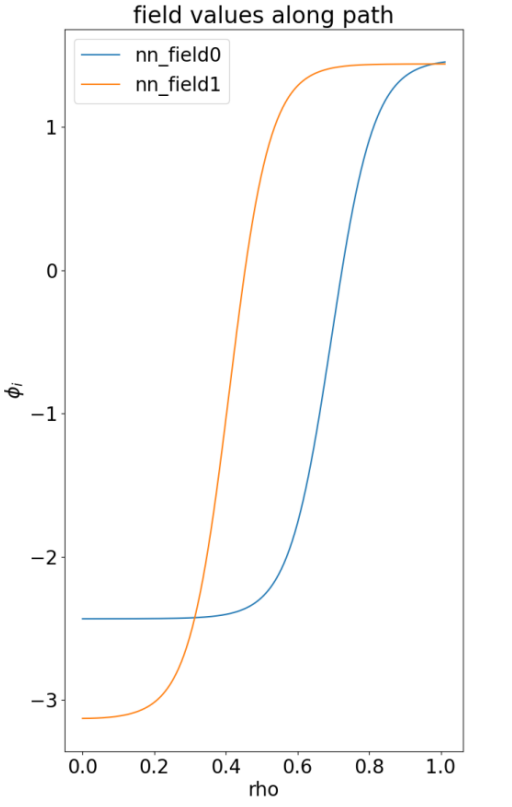
Neural network approach - Results - Toy Model

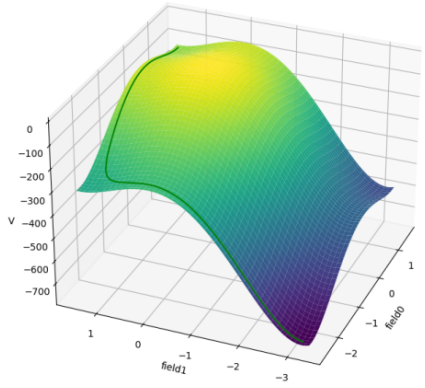
- Toy model: $V = \sum_{i=0}^{N} \lambda_i x_i^4 - A_i x_i^3 + m_i x_i^2$
- $\lambda_i, A_i, m_i > 0$ randomly generated
- Look at tunneling from highest to lowest minimum

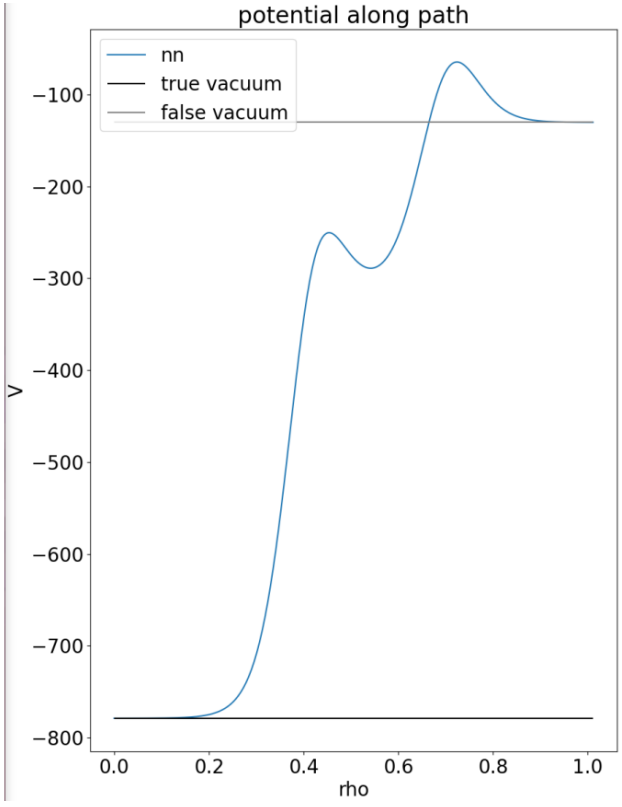
For $N = 10$ (hard limit) CosmoTransitions ran into various problems.

The network was still able to consistently provide solutions at $N = 20$.

Neural network approach - Results - Toy Model N = 2

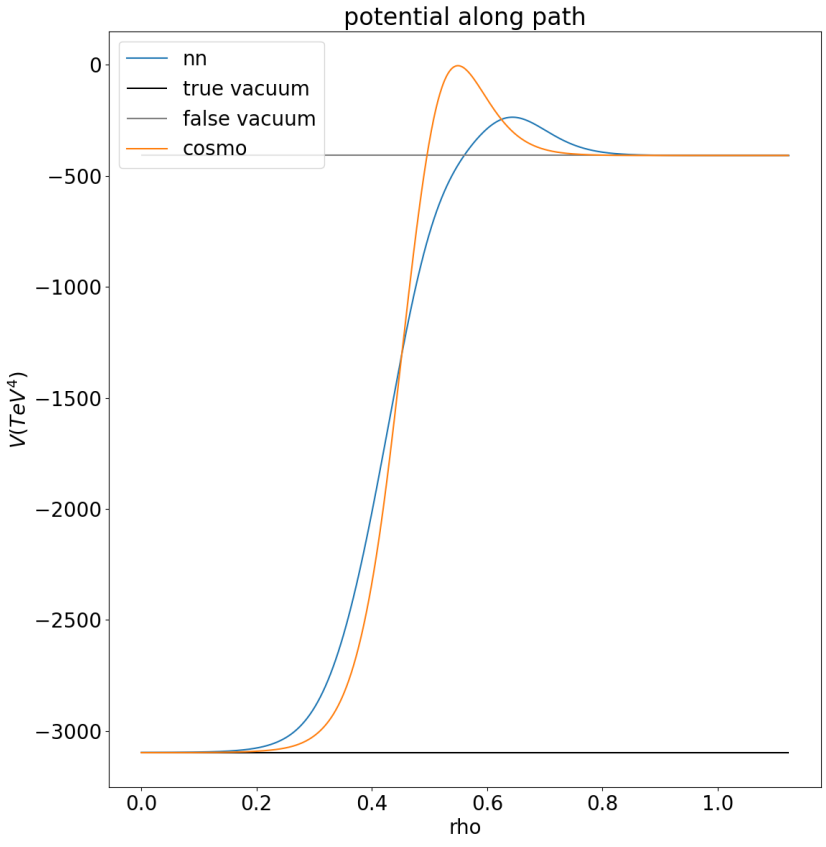


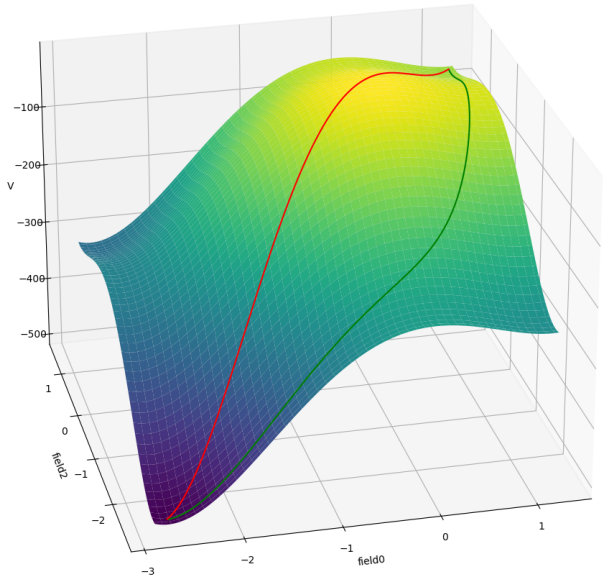




Straight: 64.68, NN: [94.92, 94.79, 94.67]
Runtime: NN on RTX 3060 Ti: 13s

Neural network approach - Results - Toy Model N = 10





Cosmo: [283.28, 286.36, 285.44], NN: [324.01, 323.67, 323.32]

Runtime: Cosmo single thread: 0.6s, NN on RTX 3060 Ti: 13s

Summary

- Added path deformation to EVADE via CosmoTransitions
- Enables large scale parameter scans with improved accuracy

- Added neural network based bounce action solver to EVADE
- Promising results, especially for cases with many scalar fields

Remaining Tasks

- Investigate effects of the various hyperparameters
- Evaluate the performance in more physics based scenarios