ENM540 Final Project Report

Multi-fidelity Finite Element Method using Conditional Variational Autoencoder

Ziyin Qu

1 Overview

1.1 Intuition

Simulation in Computer Graphics(CG) can help us achieve many realistic visual effects. Different numerical simulation techniques have been developed and applied to the industry. However, people usually takes a very long time to obtain a convincing simulation result. Also, artists have to often try and tune material parameters in order to achieve desirable visual effects and have to do the simulation many times. Undoubtedly, machine learning technique provides us some possibilities to solver this problem in a better way.

1.2 Background

The **Finite Element Method(FEM)** is a numerical method for solving problems of engineering and mathematical physics. The method yields approximate values of the unknowns at discrete number of points over the domain. To solve the problem, it subdivides a large problem into smaller, simpler parts that are called finite elements. The simpler equations that model these finite elements are then assembled into a large system of equations that models the entire problem. FEM then uses variational methods from the calculus of variations to approximate a solution by minimizing an associated error function.

Conditional Variational Autoencoder(CVAE) is an extension of Variational Autoencoder(VAE). VAE formulates the problem of data generation as a bayesian model, we could optimize its variational lower bound to learn the model. However, we have no control on the data generation process on VAE. Hence, CVAE[4] was developed to generate data with specific attribute through modeling latent variables and data.

The target of this project is to train a CVAE model with both linear and quadratic shape function FEM simulation results under different Young's modulus, so that given a certain value of Young's modulus and its corresponding linear FEM results as input, the trained CVAE model can predict the quadratic FEM results, which is

$$\{x_l, y_l\} \xrightarrow{CVAE} \{y_q\} \tag{1}$$

1.3 Related Work

With the development of deep learning, more and more powerful techniques are applied to the traditional simulation framework. In [1], a data-driven algorithm is developed to synthesis high resolution smoke simulation based on low resolution result using Convolutional Neural Networks. However, due to the naturally turbulent behavior of smoke, it is hard to tell if the add-on details are meaningful or not on smoke. So it is hard to directly apply their methods to solid simulation.

In [3], a DNN-based nonlinear deformable simulation framework is developed to reconstruct the force-displacement relation for general deformable simulation based on linear elasticity simulation. However, their model will have unrealistic deformation if the training set does not cover the feature vectors appear in the simulation.

2 Finite Element Method Derivation

For this project, we are interested in a 2D elastic beam bending problem. One side of the elastic beam is fixed to the wall, the other side is free of constraint. The elastic beam will bend under the gravity, and **neo-hookean** hyperelastic model is used to model the bending behavior. See Figure 1 for illustration.

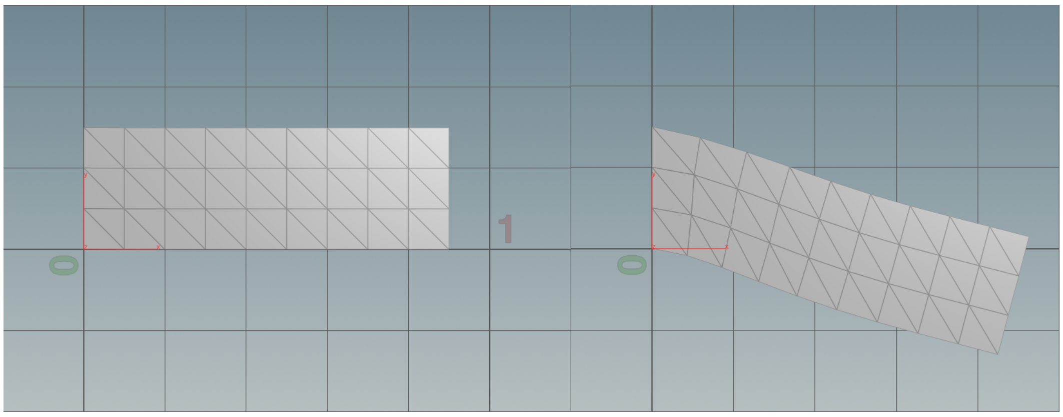

Figure 1: Elastic beam bending under gravity

2.1 Measuring deformation

Following the derivation from [2], A deformable object is characterized by a time dependent map ϕ from material coordinates X to a world coordinates x. The stress at a given point X in the material depends only on the deformation gradient $F(X) = \partial x/\partial X$ of its mapping. Since we are using a purely Lagrangian framework, all mappings are based in material space. In order to discretely represent ϕ, material space is divided into finite elements such as tetrahedrons or hexahedrons. In order to interpolate values defined on vertices in a consistent manner, we make use of isoparametric elements parameterized by ξ. The point is that nodal values of a variable x_i cam be expressed throughout the element via

$$x(\xi) = \sum_{i=1}^{n_e} x_i N_i(\xi) \tag{2}$$

where n_e is the number of vertices in the element and each $N_i(\xi)$ is an interpolating function associated with node i. Using this and the chain rule allows us to compute the deformation gradient as

$$F = \frac{\partial}{\partial X} = \frac{\partial x}{\partial \xi}(\frac{\partial X}{\partial \xi})^{-1} = \sum_{i=1}^{n_e} \frac{\partial N_i(\xi)}{\partial \xi}(\sum_{i}^{n_e} X_i \frac{\partial N_i(\xi)}{\partial \xi})^{-1} \tag{3}$$

In 2D, we can assemble the spatial positions of the element vertices in a $2 \times n_e$ matrix $D_s = [x_1, \cdots, x_{n_e}]$, and similarly for material positions, $D_m = [X_1, \cdots, X_{n_e}]$. Additionally, we assemble the derivatives $\partial N_i/\partial \xi$ in a $n_e \times 2$ matrix $H = [\frac{\partial N_1}{\partial \xi}^T, \cdots, \frac{\partial N_{n_e}}{\partial \xi}^T]^T$. With these conventions, F can be written as $F = D_s H(\xi)(D_m H(\xi))^{-1}$.

In order to estimate nodal forces for a given element, we must evaluates F at several quadrature points ξ_g. Since the element is fixed in material coordinates, $H(\xi_g)(D_m H(\xi_g))^{-1}$ is constant and can be precomputed.

For this project, we discretized our geometry into triangle elements with linear shape function and quadratic shape function. The simplest 2-D element is linear triangular element, see Figure 2 for illustration. For this element, we have three nodes at the vertices of the triangle. We introduce the natural coordinates (ξ, η) on the triangle, then the shape functions can be represented simply by

$$N_1 = \xi,\ N_2 = \eta,\ N_3 = 1 - \xi - \eta$$

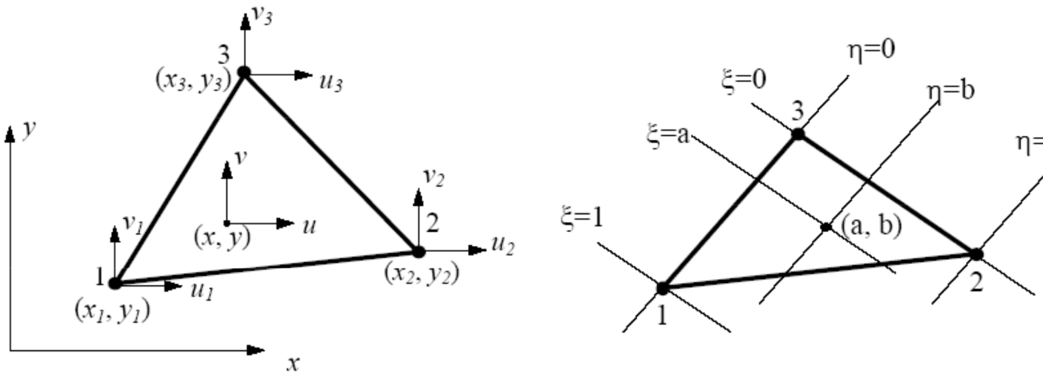

Figure 2: Linear triangular element

For quadratic triangular element, we will have six nodes: three corner nodes and three mid-side nodes. Each node has two degrees of freedom(DOF) as before. See Figure 3 for illustration. In the natural coordinate system we defined earlier, the six shape functions for the element are

$$
\begin{aligned}
&N_1 = \xi(2\xi - 1),\ N_2 = \eta(2\eta - 1),\ N_3 = \zeta(2\zeta - 1) \\
&N_4 = 4\xi\eta,\ N_5 = 4\eta\zeta,\ N_6 = 4\zeta\xi
\end{aligned}
$$

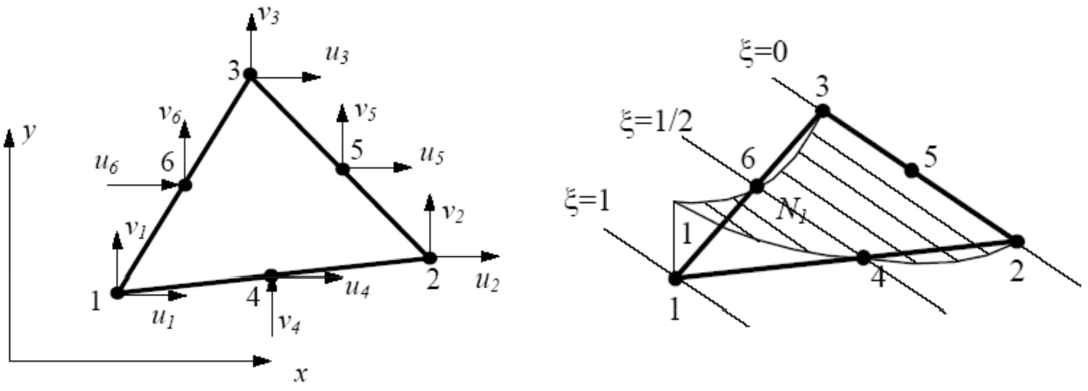

Figure 3: Quadratic triangular element

Quadratic shape functions have many advantages over linear shape functions. First of all, with the same number of elements, quadratic shape function is a higher order approximation and will have smaller error. Also, linear shape function will have shear locking problem which will lead large error for problems like bending.

2.2 Governing equation

Suppose we are using **Symplectic Euler**, we can write our governing equation as

$$\rho_0 \frac{V^{n+1} - V^n}{\Delta t} = \nabla^x \cdot P^n + f^{ext} \tag{4}$$

$$\Rightarrow \rho_0 V^{n+1} = \rho_0 V^n + \Delta t \nabla^x \cdot P^n + \Delta t f^{ext} \tag{5}$$

we rewrite this equation in the weak form, we have

$$\int_{\Omega_0} \rho_0 V_i w_i dX = \int_{\Omega_0} (\rho_0 V_i^n + \Delta t f_i^{ext}) w_i dX + \int_{\Omega_0} \Delta t \frac{\partial P_{ij}}{\partial X_j} w_i dX \tag{6}$$

where through integration by parts, we have

$$\begin{aligned}\int_{\Omega_0} \Delta t \frac{\partial P_{ij}}{\partial X_j} w_i dX &= \int_{\Omega_0} \Delta t \frac{\partial (P_{ij} w_i)}{\partial X_j} dX - \int_{\Omega_0} \Delta t P_{ij} \frac{\partial w_i}{\partial X_j} dX \\ &= -\int_{\Omega_0} \Delta t P_{ij} \frac{\partial w_i}{\partial X_j} dX\end{aligned}$$

so we can rewrite equation (6) as

$$\int_{\Omega_0} \rho_0 V_i w_i dX = \int_{\Omega_0} (\rho_0 V_i^n + \Delta t f_i^{ext}) w_i dX - \int_{\Omega_0} \Delta t P_{ij} \frac{\partial w_i}{\partial X_j} dX \tag{7}$$

if we switch the notation i, j to be α, β, we have

$$\int_{\Omega_0} \rho_0 V_\alpha w_\alpha dX = \int_{\Omega_0} (\rho_0 V_\alpha^n + \Delta t f_\alpha^{ext}) w_\alpha dX - \int_{\Omega_0} \Delta t P_{\alpha\beta} \frac{\partial w_\alpha}{\partial X_\beta} dX \tag{8}$$

In FEM Galerkin, we have

$$\begin{aligned}w_\alpha &= \sum_i w_{i\alpha} N_i^T = \sum_i \delta_{i\hat{i}} \delta_{\alpha\hat{\alpha}} N_i^T = \delta_{\alpha\hat{\alpha}} N_{\hat{i}}^T \\ V_\alpha &= \sum_j V_{j\alpha} N_j^T \\ f_\alpha &= \sum_j f_{j\alpha} N_j^T\end{aligned}$$

so we can rewrite (8) as

$$\begin{aligned}\int_{\Omega_0} \rho_0 V_{j\alpha} N_j^T \delta_{\alpha\hat{\alpha}} N_{\hat{i}}^T dX &= \int_{\Omega_0} (\rho_0 V_\alpha^n + \Delta t f_\alpha^{ext}) \delta_{\alpha\hat{\alpha}} N_{\hat{i}}^T dX - \int_{\Omega_0} \Delta t P_{\alpha\beta} \delta_{\alpha\hat{\alpha}} \frac{\partial N_{\hat{i}}^T}{\partial X_\beta} dX \\ \Rightarrow (\int_{\Omega_0} \rho_0 N_j^T N_{\hat{i}}^T dX) V_{j\hat{\alpha}} &= \int_{\Omega_0} (\rho_0 V_{\hat{\alpha}}^n + \Delta t f_{\hat{\alpha}}^{ext}) N_{\hat{i}}^T dX - \int_{\Omega_0} \Delta t P_{\hat{\alpha}\beta} \frac{\partial N_{\hat{i}}^T}{\partial X_\beta} dX \\ \Rightarrow (\int_{\Omega_0} N_{\hat{i}}^T N_j^T dX) \rho_0 V_{j\hat{\alpha}} &= (\int_{\Omega_0} N_{\hat{i}}^T N_j^T dX)(\rho_0 V_{j\hat{\alpha}}^n + \Delta t f_{j\hat{\alpha}}^{ext}) - \int_{\Omega_0} \Delta t P_{\hat{\alpha}\beta} \frac{\partial N_{\hat{i}}^T}{\partial X_\beta} dX \\ \Rightarrow (\int_{\Omega_0} N_{\hat{i}}^T N_j^T dX)(\rho_0 V_{j\hat{\alpha}} - \rho_0 V_{j\hat{\alpha}}^n - \Delta t f_{j\hat{\alpha}}^{ext}) &= -\int_{\Omega_0} \Delta t P_{\hat{\alpha}\beta} \frac{\partial N_{\hat{i}}^T}{\partial X_\beta} dX\end{aligned} \tag{9}$$

Now we switch the notation $\hat{i}, \hat{\alpha}$ to be i, α, we can rewrite (9) in the form of $Ax = b$ as

$$(\int_{\Omega_0} N_i^T N_j^T dX)(\rho_0 V_{j\alpha} - \rho_0 V_{j\alpha}^n - \Delta t f_{j\alpha}^{ext}) = -\int_{\Omega_0} \Delta t P_{\alpha\beta} \frac{\partial N_i^T}{\partial X_\beta} dX \tag{10}$$

The left hand side ($\int_{\Omega_0} N_i N_j dX$) is the mass matrix, note that we can use lumped matrix to simplify the problem. The simplest lumped mass is the row sum method, where we can rewrite our mass matrix to be

$$(\int_{\Omega_0} N_i^T dX) \delta_{ij}$$

however for triangle elements, the row sum method only work for linear case, in quadratic case, the lumped matrix will be singular, so we have to either compute the correct mass matrix, or using a different mass lumping method like diagonal scaling.

And then we can use Gaussian quadrature points to evaluate the N_i, N_j value. For linear shape function, we will need 3 Gaussian quadrature points to construct a positive definite mass matrix; For quadratic shape function, we will need 6 Gaussian quadrature points. For 3 Gaussian quadrature points, we have

$$(\xi_1, \eta_1) = (\frac{1}{6}, \frac{1}{6}),\ (\xi_2, \eta_2) = (\frac{2}{3}, \frac{1}{6}),\ (\xi_3, \eta_3) = (\frac{1}{6}, \frac{2}{3})$$
$$W_1 = W_2 = W_3 = \frac{1}{3}$$

2.3 Force computation

To calculate the right hand side of equation (10), this integral can be approximated using 3 Gaussian quadrature points as

$$\begin{aligned} &- \int_{\Omega_0} \Delta t P_{\alpha\beta} \frac{\partial N_i^T}{\partial X_\beta} dX \\ \Rightarrow &- \int_{\Omega_\xi} \Delta t P_{\alpha\beta} \frac{\partial N_i^T}{\partial X_\beta} \left| \frac{\partial X}{\partial \xi} \right| d\xi \\ \Rightarrow &- \sum_{g=1}^{n_g} P_{\alpha\beta}^g (\frac{\partial N_i}{\partial X_\beta})_g^T \left| (\frac{\partial X}{\partial \xi})_g \right| W_g \\ \Rightarrow &\sum_{g=1}^{n_g} P_g (D_m H_g)^{-T} (\frac{\partial N_i}{\partial \xi})_g^T \left| (\frac{\partial X}{\partial \xi})_g \right| W_g \end{aligned}$$

2.4 Pseudo-code

Algorithm 1 FEM simulation for 2D quadratic triangle mesh

```
1:  procedure PRECOMPUTATION($D_m$, $(D_m H)^{-1}$, $|D_m H|$, H, M)
2:      Compute and store $H_a$, $N_a$ for 3 quadrature points
3:      Compute and store $N_b$ for 6 quadrature points
4:      for each triangle:
5:          Compute $D_m$
6:          Compute and store 3 $(D_m H_a)^{-1}$ at three quadrature points
7:          Compute and store 3 $|D_m H_a|$ at three quadrature points
8:          Compute local $6 \times 6$ mass matrix using six quadrature points $N_b$ and $D_m N_b$ and $W_b$
    and add it to the system mass matrix.
9:      end for
10: end procedure
11: procedure BUILD FORCE VECTOR($f$)
12:     for each triangle:
13:         Compute $D_s$
14:         Compute 3 deformation gradient $F = D_s (D_m H_a)^{-1}$
15:         Compute 3 Fisrt Piola-Kirchhoff stress $P$
16:         Compute force and add its corresponding components to the force vector $f$
17:     end for
18: end procedure
19: Solve the system in (10) and update node velocities
20: Update node positions
```

3 Conditional Variational Autoencoder Derivation

Conditional Variational Autoencoder is a generative model. In this model, we can generate samples from the conditional distribution $p(y|x)$. By changing the value of x, we can generate corresponding samples $y \sim p(y|x)$.

Given data $\{x_i, y_i\}$, $i = 1, \cdots, n$, $x \in \mathbb{R}^d$, $y_i \in \mathbb{R}^s$, we want to learn conditional probability model $p(y|x)$, we assume that there exists some latent variables $z \in \mathbb{R}^m$, $m \ll s$, we can write our marginalized probability as

$$p(y|x) = \int p(y|x,z)p(z|x)dz \tag{11}$$

and according to the Bayesian law, we have

$$\begin{aligned} p(z|x,y) &= \frac{p(y|x,z)p(z|x)p(x)}{p(y|x)p(x)} \\ &= \frac{p(y|x,z)p(z|x)}{p(y|x)} \end{aligned}$$

3.1 Training

We want to infer $p(z)$ using $p(z|x,y)$, and thus we need to minimize the difference between the true distribution $p(z|x,y)$ and our model $q_\phi(z|x,y)$, so we can calculate the KL divergence by

$$\begin{aligned} KL\left[q_\phi(z|x,y)\,||\,p(z|x,y)\right] &= -\int \log \frac{p(z|x,y)}{q_\phi(z|x,y)} q_\phi(z|x,y)dz \\ = -\int \left[\log p(y|x,z) + \log p(z|x) - \log p(y|x) - \log q_\phi(z|x,y)\right] q_\phi(z|x,y)dz \\ = -\int \log p(y|x,z)q_\phi(z|x,y)dz + \int \log p(y|x)q_\phi(z|x,y)dx - \int \log \frac{p(z|x)}{q_\phi(z|x,y)} q_\phi(z|x,y)dz \\ &= -\mathbb{E}_{z\sim q_\phi(z|x,y)}[\log p(y|x,z)] + \log p(y|x) + KL[q_\phi(z|x,y)||p(z|x)] \end{aligned} \tag{12}$$

so we can write the evidence lower bound as

$$-\log p(y|x) \leq KL[q_\phi(z|x,y)||p(z|x)] - \mathbb{E}_{z\sim q_\phi(z|x,y)}[\log p(y|x,z)] \tag{13}$$

we will assume Gaussian distribution for the following probability,

$$\begin{aligned} p_\theta(y|x,z) &= \mathcal{N}(\mu_\theta(x,z), \Sigma_\theta(x,z)) \\ q_\phi(z|x,y) &= \mathcal{N}(\mu_\phi(x,y), \Sigma_\phi(x,y)) \\ p_\gamma(z|x) &= \mathcal{N}(\mu_\gamma(x), \Sigma_\gamma(x)) \end{aligned}$$

where their mean μ and variance Σ are the result from neural network. During the training, we will first take $\{x, y\}$ as input, pass them into the encoder neural network with parameter ϕ and obtain $\{\mu_\phi(x,y), \Sigma_\phi(x,y)\}$, then we can do the reparametrization trick to get z as

$$z = \mu_\phi(x,y) + \epsilon(\Sigma_\phi(x,z))^{\frac{1}{2}}$$

where $\epsilon \sim \mathcal{N}(0, I)$, then we can take $\{x, z\}$ as input and pass them into the decoder neural network with parameter θ to get $\{\mu_\theta(x,z), \Sigma_\theta(x,z)\}$. After we obtained all these values, we can compute our evidence lower bound and do backward pass to optimize the neural networks.

3.2 Prediction

Prediction in the Condition Variational Autoencoder is quite straightforward, given x^*, we want to the model to predict y^*, we first pass the input $\{x^*\}$ to the neural network with parameter γ and obtain the mean and variance as $\{\mu_\gamma, \Sigma_\gamma(x^*)\}$, and then do the reparametrization trick

$$z^* = \mu_\gamma(x^*) + \epsilon(\Sigma_\gamma(x^*))^{\frac{1}{2}}$$

still, we assume $\epsilon \sim \mathcal{N}(0, I)$, then we can pass $\{x^*, z^*\}$ into the deocder network and predict our result as $\{\mu_\theta(x^*, z^*), \Sigma_\theta(x^*, z^*)\}$.

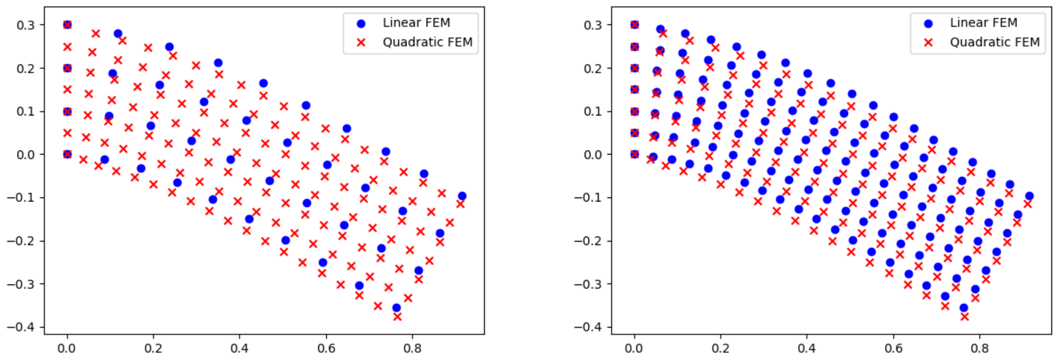

Figure 4: Data preprocessing

4 Results and Analysis

To begin with, we have to preprocess our data from the linear FEM simulation result. The reason is that for the linear FEM, we only have 6 degrees of freedom(DOF) for each triangle, however, for quadratic FEM simulation, we have 12 DOF. In order to obtain a better result, we can interpolate the linear simulation result such that it will have the same DOF as the quadratic result. Figure 4 illustrates the data before preprocessing and after preprocessing.
The training data consists of 38 linear simulation result and 4 quadratic simulation result under different Young's modulus ranged from 1000 to 100000. We assume the dimension of latent variable z is 10, 2 hidden layer of the neural network with 128 dimension. Tanh activation function is used and we use Adam optimizer to optimize the parameter of our model.

4.1 Linear to Linear

First we start with the simple case, we want to train our model so that given a certain Young's modulus, it is able to predict its corresponding linear simulation result. The training data $\{x\}$ is a 38×1 vector representing different Young's modulus, the $\{y\}$ is a 38×255 represents the linear simulation result under corresponding Young's modulus.

Because of the small size of our training data, full batch is used for training. After 20000 iterations(epoch) of training, the evidence lower bound converges as Figure 5 shows Then we com-

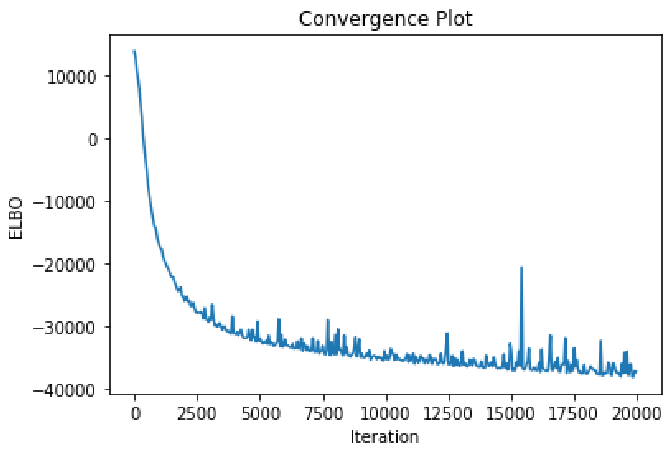

Figure 5: Evidence lower bound convergence plot

pare the model prediction with the simulation ground truth. For here, we test Young's modulus 8300, 27000, 82000. We can see from Figure 6, the model prediction matches the ground truth accurately. The model correctly learn the relationship between the Young's modulus and linear simulation result. Next, we will test the linear to quadratic case.

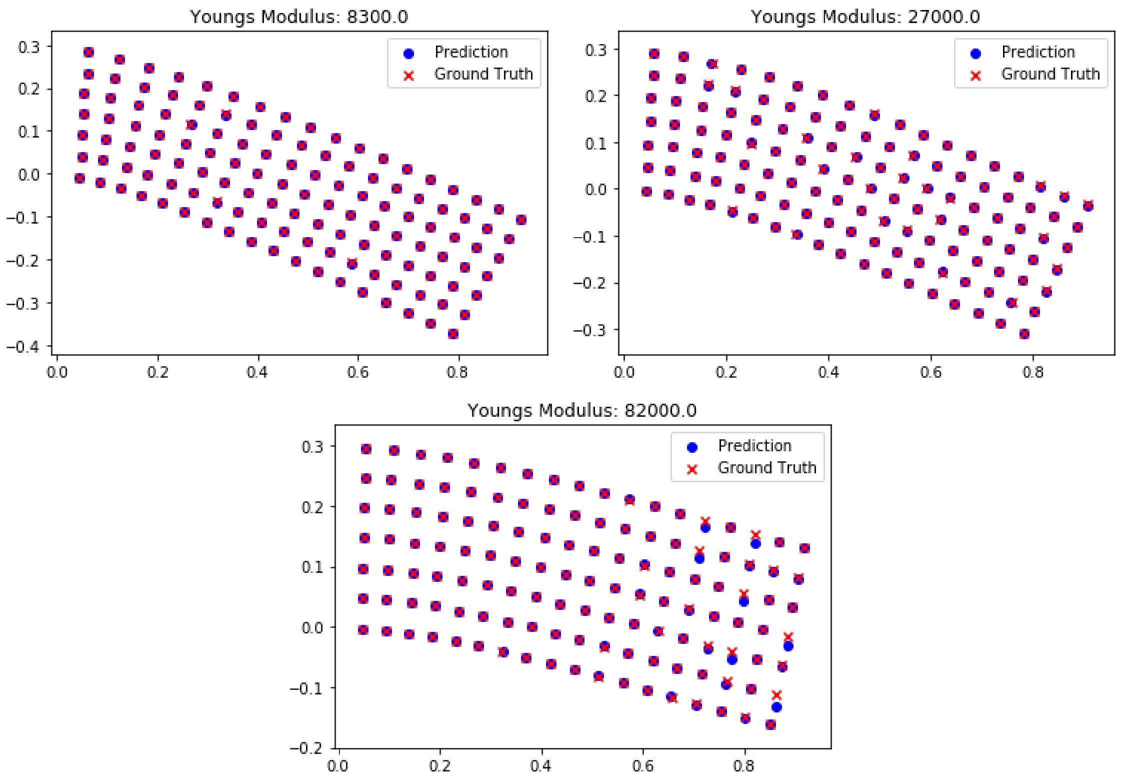

Figure 6: Model prediction and ground truth

4.2 Linear to Quadratic

For linear to quadratic, the training data is smaller, the input $\{x\}$ is a 4×255 matrix represents the linear simulation results under Young's modulus 6500, 10000, 15000, 20000. And our label $\{y\}$ is the corresponding quadratic simulation result, which is also a 4×255 matrix.

Figure 7 is the convergence plot for the linear to quadratic training. We can see that the evidence lower bound is decreasing, but is much more unstable compared with the linear to linear case.

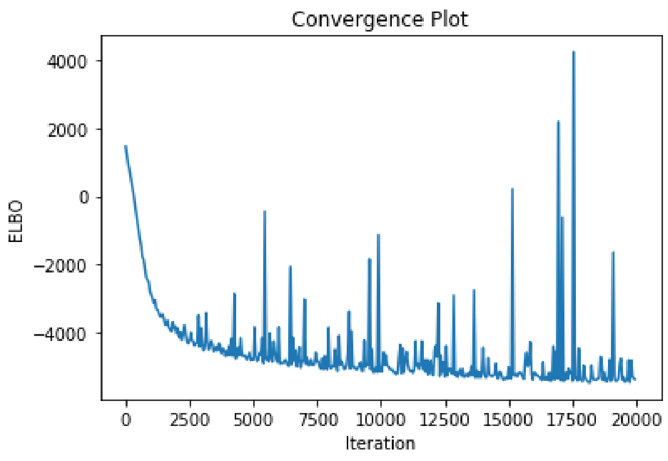

Figure 7: Evidence lower bound convergence plot

Next, we test our model by using linear simulation result under Young's modulus 5000 and 18000. As the Figure 8 shows, the model prediction is not as good as linear to linear case. Actually, the prediction for Young's modulus 18000 is close to the ground truth, it may because this Young's modulus value lies in the training data domain. It seems that the model is overfit to the training data such that it will not be able to predict the simulation result if the given input is not in between the training data. The model does not correctly learn the relationship between linear result and quadratic result.

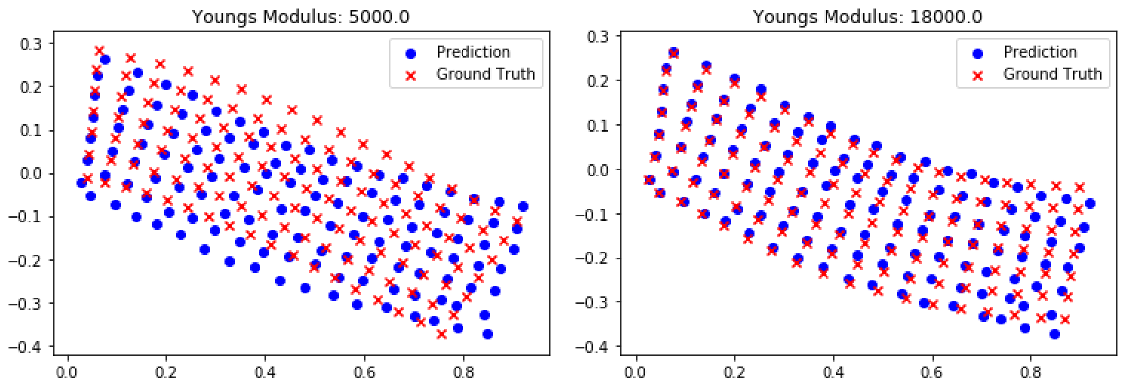

Figure 8: Model prediction and ground truth

5 Limitation and Future Work

As we can see from the results above, the Conditional Variational Autoencoder can accurately predict the linear result given a certain Young's modulus value. However, the prediction for the quadratic result is not accurate given a linear simulation result as input. So the next step for this project is to explore the reason why this model is not working well in this situation. It may because we don't have enough training data.

The limitation for this project is obvious. First of all, our model is trained to predict the simulation result for only one frame, it will be impossible for our model to predict the simulation results at different time. We can resolve this by predicting velocities instead of positions for each node and then advect our position by Explicit Euler or Runge-Kutta method. Second, our model is only trained under this simulation setting, which means if we change the boundary condition, our model will not be able to predict the result. Possible ways to solve this is that we can randomize the deformation to generate lots of training data, in this case, our model will be able to learn the relationship between linear simulation and quadratic simulation under different circumstances so that it will work under different cases.

References

[1] Mengyu Chu and Nils Thuerey. Data-driven synthesis of smoke flows with cnn-based feature descriptors. *CoRR*, abs/1705.01425, 2017.

[2] G. Irving, J. Teran, and R. Fedkiw. Tetrahedral and hexahedral invertible finite elements. *Graph. Models*, 68(2):66–89, March 2006.

[3] Ran Luo, Tianjia Shao, Huamin Wang, Weiwei Xu, Kun Zhou, and Yin Yang. Deepwarp: Dnn-based nonlinear deformation. *CoRR*, abs/1803.09109, 2018.

[4] Kihyuk Sohn, Xinchen Yan, and Honglak Lee. Learning structured output representation using deep conditional generative models. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, pages 3483–3491, Cambridge, MA, USA, 2015. MIT Press.