Data-driven Parameterization of Gravity Wave in QBO-1D model

Zihan Shao
Mentor: Prof.Edwin Gerber
NYU Courant SURE 2022

In this document, I will basically record everything worth recording of my summer research on Data-Driven parametrization of QBO, under the supervision of Prof. Edwin Gerber. We start with a brief introduction.

Introduction

Gravity Waves Parametrization

Atmospheric Gravity Waves(GWs), play an important role in the exchange of momentum between the Earth's surface and the free atmosphere. They slow down the tropospheric jet streams and stratospheric polar night jets, and are a key driver of the QBO. GWs present a challenge to climate prediction because they cannot be properly resolved with available computational power. They must rather be estimated from the larger scale variables that are resolved, or parameterized, as how it is described in the field. However, current physics-based gravity wave parameterizations neglect transient effects and account only for vertical wave propagation.[3] By using data-driven model, parametrizations will be developed to more accurately and efficiently represent GW momentum fluxes in weather and climate models. This, ultimately, will enhance our ability to predict the changes in climate and weather.

QBO-1D

Our project is located in a very interesting fact in Atomespheric Fluid Dynamics called Quasi-Biennual Oscillation, QBO for short. Equatorial zonal wind oscillates between easterlies and westerlies in the tropical stratosphere with a mean period of 28 to 29 months (quasi-biennial) due to gravity waves forcing.

In this project, in particular, we will try to do the data-driven parametrization for GWs in 1-dimensional QBO models. Followed is a mathematical characterization of the problem.

In 1-dimensional version of the problem, the unknown zonal wind u, which we expect to oscillates biannually, is governed by the following equation

$$\frac{\partial u}{\partial t} + w\frac{\partial u}{\partial z} - K\frac{\partial^2 u}{\partial z^2} = -S(u, z)$$

Here u is the unknown zonal wind, t is the time, z is the vertical coordinate, w and K are constant vertical advection and diffusivity, and $S(u, z)$ is the wave forcing term consisting of a sum of waves, which are the Gravity Waves we mentioned above[2].

Physical Parametrization with Stochasticity

For now, we have a physical parametrization of S. Currently, the forcing term is parametrized in the following way.

$$S(u, z) = \frac{1}{\rho}\frac{\partial}{\partial z}F(u, z)$$

where the wave flux $F(u, z)$ is parameterized as follows:

$$F(u, z) = \sum_i A_i \exp\left\{-\int_{z_1}^{z} g_i(u, z')\, dz'\right\}$$

and

$$A(c) = \mathrm{sgn}(c) B_m \exp\left[-\ln 2\left(\frac{c}{c_w}\right)^2\right]$$

Note that, when $z = z_1$, $F(u, z_1) = \sum_i A_i$, denoted F_{S_0}(total source flux).

The stochasticity enters the story by by making F_{S_0}(total source flux) and C_w (spectral width) Random Variables of time.

The physical parametrization, together with the PDE above, will generate a series of (u, S), which will be treated as the dataset for the data parametrization.

Our goal, in that case, is to design a Machine Learning model to predict S given (S, u) generated by the physical parametrization. Also, since we want the model to adapt to the climate change, which corresponds to different distribution of F_{S_0} and c_w, We include the randomly generated series as part of the dataset.

Goals and Evaluation metrics

Note that the evaluation metric for this problem will be different from the usual Data Science problem.

- Offline: Machine Learning model's prediction on test dataset should be satisfactory. For offline performance we mainly use (level-wise mean) / RMSE as the metric.

- Online: Machine Learning model should function similarly as the Physical Model when inserted into the PDE, with better accuracy and efficiency (hopefully). In addition, the model is expected to have the ability to generalize. Once trained on the current QBO, it should produce the actual QBO under slight perturbation of parameters(which corresponds to climate change).

In general, the Online task is of more significance, since we want the predicted model to be used in actual climate prediction.

Potential Methods

Despite human beings' great breakthrough on classification, regression is still a hard task. Here is the list of all the machine learning algorithms for regression that I can think of:

- Linear regression
- Neural Nets
- Regression trees/Forest
- Supported Vector Machine

Linear models will by no means be used in practice, we will treat it as a baseline (done in the semester before the summer). Neural nets and trees and forest have been done by other members in the group. Therefore, our main focus is the application of supported vector machine for regression(SVR for short) on our problem.

Week 1

Summer research on QBO (at least at the very beginning) can be divided into two parts. On the one hand, we are coming to the conclusion of the linear models. On the other hand, I'm starting to look at nonlinear models, such as Supported vector regression and deep learning models.

Preliminary experiments with SVR didn't even give us a reasonable result. In that case, we decided to look into the mathematical essence of SVR.

Mathematics of SVR

In this section, we will derive Supported Vector Regression in the language of mathematics.

Intuition

Like Supported Vector Machine for classification, the story begins with a hyperplane with its tube. The goal of SVM to separate the data by the hyperplane and make sure most of the data is off the tube and then maximize the diameter of the tube. Instead, the goal of SVR is to make sure most of the data will be lying in the tube and then maximize the tube. The predicted function will then be the hyperplane.

Construction

We construct the model in the following way:

- Hyperplane/predicted function:

  $$\hat{f}(x) = \langle w, x \rangle + b$$

  where $w \in \mathcal{R}^d$, $b \in \mathcal{R}^d$ where d is the number of features. w and b are trainable.

- The vertical distance between boundary, denoted ϵ. That is to say, we do not penalize the data if

  $$|\hat{f}(x_i) - y_i| \leq \epsilon$$

  ϵ is an hyperparameter.

- With this construction, the radius of the tube can be expressed as

  $$\frac{\epsilon}{\|w\|}$$

  Since ϵ is a hyperparameter, it suffices to maximize $\frac{1}{\|w\|}$. In practice, we maximize $\frac{1}{2}\|w\|^2$.

- In real applications, it is not a good idea to make every data point sits in the tube since the model would be very sensitive to outliers. Usually, we denote the upper and lower offset $\xi_i \geq 0$ and $\xi_i^* \geq 0$.
- Penalty term. To penalize the offset, we add a linear penalty term

$$C \cdot \sum_i (\xi_i + \xi_i^*)$$

to the objective function.

Optimization

Finally, we convert the problem into a optimization problem.

$$\min \frac{1}{2}\|w\|^2 + C \cdot \sum_i (\xi_i + \xi_i^*)$$

subject to $\forall i$

$$\begin{aligned} y_i - (\langle w, x_i \rangle + b) &\leq \epsilon + \xi_i \\ (\langle w, x_i \rangle + b) - y_i &\leq \epsilon + \xi_i^* \\ \xi_i &\geq 0 \\ \xi_i^* &\geq 0 \end{aligned}$$

This problem can be solved by Langrange Multipliers with K-K-T condition. Its dual problem is

$$\max L := -\frac{1}{2}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)\langle v_i, v_j \rangle - \epsilon \sum_i (\alpha_i + \alpha_i^*) + \sum_i y_i(\alpha_i - \alpha_i^*)$$

subject to $\forall i$

$$\begin{aligned} \sum_i (\alpha_i - \alpha_i^*) &= 0 \\ \alpha_i, \alpha_i^* &\in [0, C] \end{aligned}$$

Supported Vectors

With the above construction, we can derive the expression of w

$$w = \sum_i (\alpha_i - \alpha_i^*) x_i$$

Here we can see, w is in fact a linear combination of x_is. In practice, most of the coefficient will be 0. Those few vectors with nonzero coefficient will then be called supported vectors. In SVR, supported vectors uniquely determine the model.

Kernel

For now, our model can only handle linear relations. But just like what we do in supported vector machine, we map the data into higher dimensional Euclidean space so that the data has a linear relation. That is to say, if we have a map ϕ that endows the data linear relations, we can then build a SVR model on $\{\phi(x_i)\}$.

In practice, it is almost impossible to derive an effective map *phi*. However, we can take advantage of the supported vectors expression. Since in the expression of $\hat{f}$,

$$\hat{f}(x) = \sum_i \alpha_i \langle \phi(x_i), \phi(x) \rangle$$

we can neglect the actual ϕ mapping but focus on the inner product in the high dimensional space. We define the kernel function

$$K(\cdot, \cdot) = \langle \phi(\cdot), \phi(\cdot) \rangle$$

In this way, we can make the ϕ map implicit. Instead, all we need to do is to replace the inner product with kernel K in the optimization problem and the actual expression of w. The most widely used K is the Radial Basis Function, RBF in short.

Here comes the question. What kind of the kernel can be interpreted as the inner product in higher dimension. The answer is the kernel function should satisfy Mercer's Theorem.

Power of SVR

SVR has a very strong ability to approximate nonlinearity. We build SVR model on some nonlinear ground truth such as $y = \sin(x)$ or $y = \exp(-\sin x) + \cos x$, and even Trajectory of Brownian motion. All of the models approximated the function well.

Week 2

Considering Supported Vector Regression model's ability to learn non-linearity (shown by its approximation of Brownian Motion trajectory), it has the potential to reach a satisfactory performance, at least on the offline side.

Punchline: Normalization

The most basic SVR model reports terrible results($R^2 < 0$), and the fitted function is a straight line. The problem could find its origin in the objective of SVR. There's no penalty if the data sits in ϵ-tube of the data i.e. $|\hat{f}^{(k)}(u_i) -$

$s_i^{(k)}| < \epsilon$, where ϵ is set as 0.1 by default. However, we should notice that $s \ll \epsilon$, which fails the algorithm, since then a 0-line could easily optimize the objective trivially.

Since setting ϵ too small is harmful(as it would be very close to the single precision), it seems to be necessary to do normalization of the data, especially for s.

Hyperparamters Tuning

Here's the list of hyperparameters:

- K i.e. the kernel used
- ϵ i.e. the radius of the tube
- C i.e. magnitude of penalty
- the proportion of training data

We temporarily fix $K(x, y) = \exp(-\frac{\|x-y\|^2}{\gamma})$ since it is the best built-in kernel($\gamma = 73$ by default). When tuning other hyperparameters, we mainly consider the following aspects.

1. R^2 on testing data

   All hyperparameters can be influential on R^2. ϵ, in some sense, represents the degree of fitting. Before reaching overfitting, smaller ϵ and larger C will give us a better R^2. However, it is harmful to make ϵ too small. It definitely increase the training time as it takes longer to converge, but it is beyond our scope. But it increase the excution time in the mean time, which we'll discuss it later.

   Also, it may be anti-intuitive that we don't need, or we should not, pass too many training samples to the model. Despite $(0.8, 0.2)$ as the usual train-test-split, 0.2 or less training data proves to be sufficient and better for this case.

2. Excution time

   We first look into the predicting function of SVR model. Given u, the prediction comes from

   $$\hat{f}(u) = \sum_i \alpha_i K(u_i, u) + b$$

   As we can see from the expression, excution time depends on

- # of supported vectors. Note that in our case(and also in most of cases) K is a nonlinear term, thus it is not possible to merger all $\{u_i\}$s. Therefore, the number of supported vectors(where α_i nonzero) is positively co-related to the excution time.
- Dimension of u. As inside the kernel function, we need to compute the inner product of $u - u_i$, shorter u will results in shorter runtime.

It is straightforward to control # of supported vectors by ϵ. Small ϵ will lead to a strict tube resulting in more supported vectors, and vice versa. Usually, we control # of supported vectors between 500-1500.

As for dimension of u, we can do dimension reduction on u. We begin with the most basic dimension reduction method, Principle Components Analysis, PCA for short.

PCA

We omit the algorithm of PCA here. But PCA is definitely something worth trying as it is a linear transform that has no cost.

Surprisingly, when we insert u into the PCA model, the first two principle components account for 0.97 of the importance. This seems to be a good news, as then the input data can be

$$(\hat{u}_1, \hat{u}_2, s_f, c_w)$$

But the result is not good, but dimension reduction is still possible. We finally keep 16 dimensions of the u.

Problems with SVR

Bad Online Testing Result

For now, the online testing of the SVR model does not reveal the correct SVR.

Slow

SVR, in general, has a really uncontrolable testing time. It takes several minutes to run the PDE solver for 96 years.

One possible solution is to rewrite the kernel function. Now the kernel needs to compute the exponential function, and it is reasonable to use a self-designed kernel only using the first 3 terms of power series of exponential function.

Plan for next week

1. Modify the initial condition of the online testing. Use the last slice of the control run to be the initial condition for online testing.
2. Implement the RBF kernel manually in a (hopefully) more efficient way. Current idea is to only use the first 3 terms of the Taylor series.
3. Keep trying dimension reduction. Try if we can further reduce the dimension with PCA or other more advanced dimension reduction methods.

Week 3

Online testing with different initial condition

We used the last slice of the control run to be the initial condition of the zonal wind and hope it will lead to a good result of the SVM model. However, it still didn't give us a good result. Part of the reason is because the model is still too sensitive.

Self-designed kernel

I have tried to self implement the Radius basis kernel keeping only the first 3 elements. The results seem not to be good.

Relevance Vector Machine

The experiment shows that perhaps Supported Vector Machine is not a good idea for our project. The predicting speed limits it from being the method that we would like to choose. (Though in the sense of proof of concept, it still preserve a chance to reveal a reasonable online testing performance).

In comparison, Relevance Vector Machine sounds more reasonable for our need. One biggest difference is Relevance Vector Machine relies on considerably less supported vectors, thus being unsuitable for our project.

As usual, we then give a rigorous introduction of Relevance vector machine.

Disadvantages of Supported Vector Machine

The story begins with the disadvantage of Supported Vector Regression.

- SVR has no statistical interpretation
- Though sparse, SVM makes liberal use of Kernel functions.
- Hyperparameters require cross-validation

- Kernel function must satisfy Mercer's Theorem

The Relevance Vector Machine utilize the functional form of the SVM, and is a probabilistic sparse kernel model. In short, the essence of the model is Bayesian Inference.

Bayesian construction

We start with the predicting function of supported vector machine. Given input $\{x_n, y_n\}_{n=1}^N$, we have

$$y(x) = \sum_{n=1}^{N} w_n K(x, x_n) + w_0$$

Recall what we did for least square estimation, we want to maximize the conditional probability of $w = [w_0, w_1, \ldots, w_n]^T$ given the data. This is a posterior distribution and we construct prior and likelihood as normal distribution (as they form a conjugate prior).

- Likelihood:

$$\mathbb{P}(y|w) = \mathbb{P}(y|w, \sigma^2) = (2\pi\sigma^2)^{-N/2} \exp\{-\frac{1}{2\sigma^2}\|y - \Phi w\|^2\}$$

  where $y = [y_1, y_2, \ldots, _n]^T$, Φ is the $N \times (N+1)$ matrix, $\Phi_{nm} = K(x_n, x_m - 1)$, $\Phi_{n1} = 1$. Note that here σ^2 is a hyperparameter.

- Prior:

$$\mathbb{P}(w) = \mathbb{P}(w|\alpha) = \prod_{i=0}^{N} (2\pi\alpha_i^{-1})^{-1/2} \exp\{\frac{1}{2}\alpha_i w_i^2\}$$

The posterior likelihood can be derived from the product of likelihood and prior. However, the calculation is highly nontrivial. Since we only cares about the proportion, we can simplify the calculation thanks to the following property.

- The product of two normal distribution((μ_1, Σ_1), (μ_2, Σ_2)) is proportion to the density of a new normal distribution, whose parameters are given by

$$\Sigma = (\Sigma_1^{-1} + \Sigma_2^{-1})^{-1}, \mu = \Sigma\Sigma_1^{-1}\mu_1 + \Sigma\Sigma_2^{-1}\mu_2$$

Therefore, we have the posterior

$$\mathbb{P}(w|y) = (2\pi)^{-(N+1)/2} |\Sigma|^{-1/2} \exp\{-\frac{1}{2}(w - \mu)^T \Sigma^{-1} (w - \mu)\}$$

with

$$\Sigma = (\Phi^T B \Phi + A)^{-1}$$
$$\mu = \Sigma \Phi^T B t$$

where $A = \text{diag}\{\alpha_0, \ldots, \alpha_N\}$ and $B = \sigma^{-2} I_N$

With the posterior, we are able to get the maximum likelihood estimator. However, the hyperparameters σ^2 and α are unknown.

Marginal distribution, Evidence

To find the optimal hyper-parameter, we introduce the evidence, or marginal likelihood. Basically speaking, it is the posterior distribution with w integrated out. The evidence means the probability of that data happens given the hyper-parameter. With some computation, we have

$$\mathbb{P}(y|\alpha, \sigma^2) = (2\pi)^{-N/2} |B^{-1} + \Phi A^{-1} \Phi^T|^{-1/2} \exp\{-\frac{1}{2} y^T (B^{-1} + \Phi A^{-1} \Phi^T)^{-1} y\}$$

Given the nature of the expression, it is natural to find the hyperparameters α, σ that maximizes the probability. Then it suffices to do make the prediction with the posterior distribution.

Optimization of hyperparameters

As usual, we inspect the log-evidence function. Let $C := (B^{-1} + \Phi A^{-1} \Phi^T)$

$$\begin{aligned} L(\alpha, \sigma^2) &:= \log \mathbb{P}(y|\alpha, \sigma^2) \\ &= -\frac{N}{2} \log(2\pi) - \frac{1}{2} \log |C| - \frac{1}{2} \log(y^T C^{-1} y) \\ &= -\frac{1}{2} (N \log(2\pi) + \log |C| + \log(y^T C^{-1} y)) \end{aligned}$$

We want to zero out the partial derivative w.r.t α and σ. In a nutshell, we can optimize α and σ^2 iteratively by the following recipe

$$\alpha_i^{\text{new}} = \frac{\gamma_i}{\mu_i^2}$$
$$(\sigma^2)^{\text{new}} = \|y - \Phi \mu\|^2 / (N - \sum_i \gamma_i)$$

Sparsity

During re-estimation, we may find that many of the α_i goes to infinity. Consequently, the posterior becomes infinitely peaked at zero, which implies that the corresponding kernel functions can be pruned. Intuitively, if $alpha_i$ goes to infinity, the variance of the prior goes to 0. Namely, there's no information contained in w_i, thus being wiped out.

No-cost Improvement

As we can see, the Relevance Vector Machine starts with the functional expression of Supported Vector Machine. It thus can be regarded as a no-cost improvement of Supported Vector Machine, except for the increased training time, which is beyond the cope of our project.

Experiments with Relevance Vector Machine

Some preliminary trials with Relevance Vector Machine shows us the following

1. The Run time of Relevance Vector Machine is indeed very big.
2. For now, Relevance Vector Machine suffers from severe overfitting.
3. Both Supported Vector Regression and Relevance Vector Regression shows difficulty in learning source term in high level, especially for Relevance Vector Machine.

We should be aware that, in the following experiments for SVR and RVR

- SVR is slow, so the work is focusing on the proof of the concept. We want to see, given no limitations on run time (thus no limitations on numbers of supported vectors), whether or not he SVR model can reflect reasonable performance on online testing.
- RVR is promising. Its run time still preserve a chance for us to get a model that is comparable to Neural nets.

Week 4

This week stats from some boring stuff. Since local computer cannot handle the great computation of RVR training, we then are in need of the aid of HPC.

It is no trivial...In fact it is a very tough process.

After 3 days I finally get the HPC working correctly. Then we come to experiments.

Experiments with SVR

Objective

Though the predicting speed of SVR blocked itself from becoming the practical model, we still want to do the proof of the concept **i.e.**, given no restrictions on the number of supported vectors, is it possible for the SVR model to produce a reasonable online testing result.

Hyper-parameters

First of all, we specify the hyperparameters that we want to adjust.

The first two hyperparameters are ϵ and . Generally speaking, making ϵ smaller and C bigger can give us better offline performance. Also, making ϵ smaller will definitely give us more supported vectors.

There's another hyper-parameter, which can be found implicitly in the RBF kernel

$$K(x, y) = \exp(-\frac{1}{\gamma}\|x - y\|^2)$$

Note that, the x and y we input into the kernel function is the augmented u. However, the unit of u and s_f and c_w are different (which requires normalization in the first place). We could then add a coefficient before s_f and c_w manually, denoted K, since the model doesn't have the ability to adjust the weight itself. That is to say, we have the input data U(augmented)

$$U = [u_1, \ldots, u_{73}, K \cdot s_f, K \cdot c_w]^T$$

Note that, we still neglect a lot of hyperparameters here. For example, we do not do the dimension reduction at this point. Also, we are currently using the same hyperparameters for all levels. However, the difference between levels are nontrivial, for example, it may be easier to learn the lower level source term than higher levels. however, the sklearn package only supports uniform parameters for now.

Grid Search

Based on the 3 parameters, we conduct a gird search. Overall, we tested 120 models, with hyperparameter ranging from

- $\epsilon \in \{0.50, 0.10, 0.05, 0.02, 0.01, 0.005\}$
- $C \in \{8, 16, 32, 64\}$
- $K \in \{0.5, 1, 2, 4, 8\}$

and we record

- Offline R^2/loss on lower, middle higher level
- Number of supported vectors on lower, middle higher level
- Offline performance: R^2/loss
- Online performance, including QBO parameters(amplitude and period), and QBO objective, which is given by the expression

$$\frac{(\sigma_{25} - 33)^2}{33^2} + \frac{(\sigma_{20} - 19)^2}{19^2} + \frac{(\tau_{25} - 28)^2}{28^2}$$

- Online testing time(duration)

Result of Grid Search

We very briefly describe the result here and leave the thorough analysis to next week.

We tested 120 models. Around 60 of them has the correct online performance. Generally speaking, the one with higher offline performance(higher R^2) has a bigger probability of satisfactory offline performance. That is to say, given no restrictions on the number of supported vectors, a smaller ϵ will make the online testing more promising.

However, the number of supported vectors are far to big. The average of supported vectors may be around 4000.

Generalization

We also tested our model with slight perturbation(biased mean, biased variance, anti-correlation). We tested 45 well-performed model. According to our experiments, most of the model can qualitatively get biased variance and anti-correlation correct. Very few of them can handle biased mean, even in a qualitative sense.

Week 5

Simple Analysis

First of all, we do some analysis on the stats we have. We have the following rough conclusions

1. K and C have no evident impact on the online testing performance. We can wait and see K's impact on the ability of generalization.
2. Generally speaking, the offline performance are positively related to the probability that a model has a reasonable online performance.

The next goal is to find out the minimal number of supported vectors.

Number of Supported Vectors

The first question is, is the number of supported vectors the key factor.

The idea is the two model can have similar numbers of supported vectors with different causal effect. There are mainly two factors, one is the ϵ and the other is `test_size`. In particular, given ϵ the `test_size` linearly determine the number of supported vectors.

We do experiments on different ϵ and `test_size`.

- $\epsilon \in \{0.0025, 0.001, 0.0005, 0.0001\}$
- `test_size` $\in \{0.80, 0.81, \ldots, 0.99\}$

Based on the result, we could see the number of supported vectors are not determining. How the coefficient are optimized are also important. Our results show, for each ϵ, 5% of the data are needed for a successful online performance. The minimal supported vectors is then with a average of 1500 (which is absolutely better than what we conjectured before).

Also, this observation gives us some confidence on the reletive vector regression. Since the number of supported vectors are important but how they are optimized are also important.

Amount of Training Data

We first claim the following: Though there are a lot of factors affecting the efficiency of the model, what we care at this point is the number of trainable parameters.

The trainable parameters for supported vector are quite easy to compute. All in all, what is up to be determined is the coefficient of the vectors(either zero of nonzero). Therefore, the number of trainable parameters are(ignore the bias term):

$$\#\text{trainable parameters} = \text{output dimension} \times \#\text{training data}$$

Therefore, based on our experiment, the number of trainable parameters have a lower bound of $71 \times 1500 = 1 \times 10^5$

Optimize the Engineering

What is interesting is, the number of supported vectors seems to be irrelevant to the online duration. Theoretically, the prediction is made based on

$$\hat{f}(x) = \sum_n \alpha_n K(x, x_n) + b$$

It can be written into the matrix form. Given the multiregressor, we can have direct acess to b and the weights α_i. And it remains to calculate the basis matrix which is given by

$$B_j = K(x, x_j)$$

However, the prerequisite of this fast prediction is the supported vectors should be relatively dense.

Based on the experiments, this method is efficient when the number of supported vectors are small. Generally speaking, the seemed-promising method is failing.

A Way Out: Relevance Vector Machine

As we have mentioned before, relevance vector machine is something worth trying. It utilize the functional expression of the Supported Vector Machine and combine it with Bayesian Statistics. We hope it could give us a better online performance with improved efficiency.

Offline Performance

Training of the Multi-Output RVR model is not hard. Thanks to its improvement on SVR, we have only two hyperparameters to decide:

1. K, the coefficient for s_f and c_w when combined with u. Since in the previous experiment, K is not significant. We set it to 1 at this point.
2. `test_size`. As I have shown before, this hyper-parameter uniquely determined the trainable parameter of the model.

During training, we encounter several problems and solve it accordingly:

1. Error encountered when training the first and the last layer(Due to taking inverse). Since they are zero, we skip the training at this point and manually marked them zero.
2. Long training time. The training time for one dimension output is around 4-8 minutes. Therefore, the total training time, without parallel computing, is 71×6 min(7 hour), which is unacceptable. Therefore, we train the model parallelly(actually, submit a batch job for each dimension) and then combine them together.

We have the following result

1. RVR model, compared with SVR model, gives us a significantly better result: Higher R^2 score, with fewer Supported Vectors (# denotes the mean relevance vectors).

   `test_size` $= 0.95$, $R^2 = 0.92$, $\# = 298$

   `test_size` $= 0.90$, $R^2 = 0.95$, $\# = 422$

   `test_size` $= 0.80$, $R^2 = 0.96$, $\# = 493$

   `test_size` $= 0.50$, $R^2 = 0.97$, $\# = 375$

   `test_size` $= 0.25$, $R^2 = 0.97$, $\# = 473$

2. Interesting facts
   (a) Number of relevance vectors used is not closely related to the amount of training data.
   (b) Generally speaking, the more the training data offered, better offline performance it would have.

Online Performance

The Online Performance of the RVR model is terrrrrrrrrrrrrrible, which is out of our previous expectation. For the five model listed above, none of them reports reasonable QBO parameters. Also, the execution time is unexpectedly long. The reason is unknown, as the execution time of RVR model should be significantly shorter then SVR model due to fewer supported vectors/relevance vectors used.

Possible reason for its unsatisfactory result is, though RVR models acheived high online performance, it is essentially a light-weighted model, which means it's not stable under small perturbation.

We are discouraged by the fact that RVR model, after a huge effort for implementation, may not be a good choice. However, future effort can be made. For example, for a proof of concept, can we make the model work by increasing the amount of training data.

Possible way out

We are going to try

1. Proof of concept: given no restrictions, does the RVR model have the ability to have a reasonable online performance.

   We thus tried the following:

   `test_size` $= 0.25$, $R^2 = 0.97$, $\# = 473$

   `test_size` $= 0.10$, $R^2 = 0.97$, $\# = 508$

   Both of them fail in online testing.

2. If the RVR model cannot do the Online task, can it help the SVR model have a better performance? For example, as we can see, some levels of the SVR model has bad offline performance. Since the RVR model and the SVR model has the same predicting functional structure, we can replace it with the model to see if it can performs better.

3. Control the offline performance. SVR has already shows that, we don't need a perfect offline performance to achieve a reasonable result in Online performance. We may manually lower the R^2 to make the model better. We can raise the threshold for α(the dual-coef for relevance vectors).

Week 6

Not much progress made in this week due to some personal trifles. We first respond to the potential remedies mentioned in the last section.

Remidies for Relevance Vector Machine

Based on our conjecture, the reasons why the RVR fails in Online testing is the insufficiency of the number of vectors used in predicting, thus making the model unstable. Therefore, the motivation for all trials is based on increasing the number of relevance vectors. The naive idea is to increase the number of supported vectors to 1500 then it should be working online(as it outperform the SVR in terms of offline, thus it should work with the same number of supported vectors).

However, in RVR models, we lose the control of # due to deprivation of hyperparameters. Generally speaking, we only have 2 ways of increasing the number of supported vectors.

1. Increase the size of training dataset
2. Adjusting the threshold for α (coefficient) so that more vectors can be included.

However, none of them increases the number of Relevance Vectors effectively, which means

1. The number of relevance vectors is invariant of the size of input. (In the case of Supported Vector Machine, the number of supported vectors is proportional to the size of input data).
2. The number of relevance vectors doesn't increase a lot given a raised threshold (We change `threshold_alpha` = $1e5$ to `threshold_alpha` = $1e8$). It reveals that those vectors pruned has a relatively small impact on the model.

These result shows that the relevance vector machine is a really robust model in terms of the offline task. However, we fails to show the number of relevance vectors is the key factor. It might be, but we have insufficient evidences for now.

Week 7

For now, all our effort is based on the method itself. We neglect the input data (except for the `text_size`). Now we want to make the data a little bit more complex to see if it can help the model to do online testing. Let's begin with why we need to do that.

The need for complex data

The idea comes from the difference between `deterministic forcing` and `stochastic forcing`. It seems that it is easier for models to do the online emulation if fed `stochastic forcing` data. We guess it is because the `stochastic forcing` data let the model to see a wider track of oscillation.

Ways to add complexity

Annual cycle for w, K

Since sf and cw are already stochasticaly generated, we first focus on w and k, which are two constant in the equation.

$$\frac{\partial u}{\partial t} + w\frac{\partial u}{\partial z} - K\frac{\partial^2 u}{\partial z^2} = -S(u, z, A, c)$$

Currently, we are using constant w and K. We can try add some complexity by making these two hyper-parameters periodic. However, Ofer told me, based on his experiments, complexity remains the same with annual cycle of w and K.

More Randomness for sf and cw

Currently, sf and cw are generated randomly with given mean and variance. What we are going to do is to perturb the sf and cw in a continuous way. It is clear that we should not include the perturbed data during the training since that will be considered as cheating.

However, it may be legitimate to perturb the sf and cw continuously. In practice, we do the following.

1. Sample sf and cw stochastically with given mean and variance.
2. We want do the perturbation by the following scheme

$$sf_{new} = k_{sf} \cdot sf + b_{sf}$$
$$cw_{new} = k_{cw} \cdot cw + b_{cw}$$

   Where the k change the variance (and mean) and b changes the mean. We don't want to pay too much attention on how k and b changes the mean and variance. The rough idea is to generate a complex data
3. In practice, we generate an linear array of k and b, whose length is the same as sf and cw. Then we multiply the k and add b entrywise.

On the one hand, the data include implicitly the perturbed data in some sense, even though just one glimpse for each perturbed case. On the other hand, the generated u may (should) not represent any regular QBO. In that case, we could say the extra information for how to generalization is compensated by not seeing a standard QBO, which makes it a fair game.

When doing the online testing, we will insert the regular sf and cw. We expect the model can learn the physics, thus generating the correct QBO.

Goals

Explicitly, we expect the complex data to

- test if the SVR model is really learning the physics. If the model can still generate the correct QBO, we say it is learning the true physics as it never see the standard QBO when training.
- help the model to generalize, as it is seeing changing sf and cw when training
- help RVR model to succeed in Online testing.(It fails on the standard dataset despite a very good offline performance)

Experiments with SVR

Scheme

We sample k and b with the range

$$\begin{aligned} k_{sf} &\in [1, .1] \\ b_{sf} &\in [0, 1e-3] \\ b_{cw} &\in [0, 10] \end{aligned}$$

and do the training and online testing with 120 sets of hyperparameters used before.

Result

Several conclusions can be drew from the result of the experiment.

1. It is easier for the model to learn the complex data (R^2 is higher than before).
2. With the more complex data, it is easier for the model to do the online task.(For the model with the same hyperparameters, sometimes the one trained on the complex data can achieve a reasonable oscillation, while the one trained on the standard data cannot)

3. For the model trained on the complex dataset, its QBO statistics are not as accurate as before (given that both models works online)

4. Complex dataset enhance the models' ability of generalization.

Note that, conclusion 3, and 4 are quite intuitive. Since the model didn't see the correct QBO during training, it is harder for the model to get the QBO statistics correct. However, as it has seen a wider range of sf and cw, the model naturally has a better ability to generate.

Experiments with RVR

We hope the complex data can help the RVR to succeed in Online training. We tried the model with `test_size` $= 0.20$, $R^2 = 0.98$, $\# = 565$ However, it still fails the Online testing.

Week 8

Sweep `cwe` and `sfe`

To investigate more about the ability to generalize, we try to plot the sweeping sfe and cwe (Ofer adopted it to visualize the ability to generalize).

Followed is the result between the best model with regular data (model 1) and the best model with complex data (model 2).

1. Model 1 fails easily on some boundary cases.(Therefore, the picture may not reveal its ability to generalization). We are going to try a smaller range.

2. Model 2's picture is very clear. But this might be due to a wider range of supported vectors.

3. Generally speaking, model 2 has a better ability of generalization. Its tendency is qualitatively correct (compared with the ground truth). What is surprising is, its result is even better than the result by NN.

Different motivation generating of complex data

There are several different mindset for generating complex data.

1. Previously, we are using a point to train the model. However, we can feed the model with an environment.

2. Let the model to learn the physics.

I suggest to use a curve to generate the complex data.

RBF kernel revisit

Prior to that, we have not investigated RBF kernel function. There's only one hyperparameter up to determine, which is γ. It is the coefficient in the RBF kernel. We use the expression:

$$K(x, y) = \exp(-\gamma \|x - y\|^2)$$

Meaning of γ

We consider the zero-centered rbf kernel function. The value of γ present the degree of localization. If we set a larger γ, the RBF kernel is thus more centered.

In another perspective, we can view the RBF kernel as the Laplacian distribution. Then the predicting function is the mean of the random variable.

But that goes too far. What we need to know for now is, a larger γ makes the optimization stricter, which results in more supported vectors. This idea is initially to improve the performance of Relevance Vector Machine. When we make γ big, the number of supported vectors will increase, which might largely enhance the online performance of RVR models.

Grid Search of γ on RVR models

The grid search on γ does not give us a satisfactory online performance. This disprove the hypothesis that an certain amount of supported vectors should be reached to achieve a satisfactory online performance.

Grid Search of γ on SVR models

Here are two different motivations to modify γ in SVR models, both of which with the essential goal to minimize the number of supported vectors required for a satisfactory online performance.

1. Making γ small so the model will take fewer supported vectors.
2. Making γ big so the model will be more strict. This will result in more supported vectors. However, the number of supported vectors is bounded by the size of training dataset.

Again, a larger γ will make the approximation more accurate. Therefore, (1) doesn't make too much sense.

With (2), we are going to largely improve the number of supported vectors required for a satisfactory online performance. Here's some results.

1. Even the one with only 0.5%(172 supported vectors) of training data can achieve good QBO parameters. But all of them don't have the ability to generalize.

2. The model with 800 supported vectors started to have the ability to generalize.

3. Based on the stats, $\gamma = 0.028$ is the optimal(we use $\gamma = \frac{1}{75} = 0.013$ before).

Week 9

In this week, the main focus is to wrap up the works before. But still, we have some interesting findings.

Extrapolation and Interpolation

I'm trying with a more reasonable complex dataset with changing mean and variance. It is like a curve centered at the default value of sfe and cwe(using a sin and cos for generation). This dataset works well with perturbation. From my point of view, that's still kind of extrapolation as the model didn't see the distribution outside of the covering of the curve.

But in another sense, the model sees how to react with different distribution of sf and cw, which make the perturbation an interpolation for it.

A (potential) complex dataset

Other than a more complex sampling of sf and cw, we could add complexity on w. The idea is to set a annual cycle of w. Note that u is quasi-biannual oscillating, which leads to the phase difference.

There are two ways of doing it:

1. Train on dataset with annual cycle of w and do the online emulation with **constant** w.

2. Train on dataset with annual cycle of w and do the online emulation with **annual cycle of** w **as well**.

Both of the results shows that the second one does not make a significant difference.

Week 10

In this week, we finally nailed to some specific models (for the paper). Here are the hyperparameters that we use for both regular data and more complex data with annual cycle of w.

Also, the most important three things in this week are: Mom's Birthday, Presentation, and my Birthday (finally I can drink legally haha).

Regular data

For the regular data, we have two models, one is a full-sized model, and the other is a light-weighted model. Here are the statistics.

Physical model's online statistics:

$$\begin{aligned} \tau_{25} &= 27.4 \\ \text{amp}_{25} &= 33.4 \\ \text{amp}_{20} &= 18.5 \end{aligned}$$

Full-sized

Hyper-parameters:

$$\begin{aligned} \epsilon &= 0.0001 \\ \gamma &= 0.05 \\ C &= 16 \\ K &= 1 \\ \texttt{test\_size} &= 0.80 \end{aligned}$$

Results:

$$\begin{aligned} R^2 &= 0.95 \\ \#SV &= 6670 \\ \tau_{25} &= 27.4 \\ \text{amp}_{25} &= 33.4 \\ \text{amp}_{20} &= 18.5 \end{aligned}$$

This full-sized model is very accurate. Besides, it also generalizes well, except it inevitably blows up at the corner of the sweeping grid. However, it is even quantitatively correct locally.

Light-weighted

Hyper-parameters:

$$\begin{aligned} \epsilon &= 0.0001 \\ \gamma &= 0.04 \\ C &= 16 \\ K &= 4 \\ \texttt{test\_size} &= 0.99 \end{aligned}$$

Results:

$$\begin{aligned} R^2 &= 0.67 \\ \#SV &= 334 \\ \tau_{25} &= 28.03 \\ \text{amp}_{25} &= 30.9 \\ \text{amp}_{20} &= 17.0 \end{aligned}$$

The light-weighted model is not good at the accuracy.

Data with annual cycle of w

With the annual cycle of w, it becomes a harder problem to accomplish. SVR's performance is still satisfactory but lack in exactness. Generally speaking, the models are still accuate in terms of amplitude, but fails to capture the correct period. The possible reason (very possible) is, due to the difference in phase speed of w and u, the period is essentially a harder task now.

Week 11

This week we started to investigate SVR with MiMA data. At this point, we first checked its offline performance.

The offline performance is nontrivial. The dataset we have is as followed.

- Zonal Wind: $\texttt{u}(\texttt{t}, \texttt{latitude}, \texttt{longitude}, \texttt{pressure})$
- Gravity Wave: $\texttt{s}(\texttt{t}, \texttt{latitude}, \texttt{longitude}, \texttt{pressure})$

Currently, there's no stochasticity the model, which means we have no sf and cw to include.

Structures for Offline

To insert into the SVR model, we need to flatten the 4-dimensional data to 2 dimensional data. However, We have 2 options.

option 1

We transform the data into

- Zonal Wind: $\texttt{u}(\texttt{t} \cdot \texttt{latitude} \cdot \texttt{longitude}, \texttt{pressure})$
- Gravity Wave: $\texttt{s}(\texttt{t} \cdot \texttt{latitude} \cdot \texttt{longitude}, \texttt{pressure})$

We retrieve the pairs of zonal wind and gravity wave at each time step and each location. We do not distinguish the location that the zonal wind is retrieved.

In this sense, the machine learning function generated will be

$$\hat{f} : \mathbb{R}^{40} \to \mathbb{R}^{40}, \quad f(\mathtt{u}[\mathtt{i}, \mathtt{j}, \mathtt{k}, :]) = \mathtt{s}[\mathtt{i}, \mathtt{j}, \mathtt{k}, :]$$

option 2

We transform the data into

- Zonal Wind: $\mathtt{u}(\mathtt{t}, \mathtt{pressure} \cdot \mathtt{latitude} \cdot \mathtt{longitude})$
- Gravity Wave: $\mathtt{s}(\mathtt{t}, \mathtt{pressure} \cdot \mathtt{latitude} \cdot \mathtt{longitude})$

We retrieve the pairs of zonal wind and gravity wave at each time step while keeping the longtitude and latitude. That is to say the output will be a function from a matrix to a matrix.

It seems that precarious to do this. However, it is very easy to make some sense. On the one hand, in our 1-d case, what we did is several independent 1-d regression. When it comes to a rectangular matrix, the idea is presevered. Also, in terms of the input, the input are also independent in some way. Different position in the matrix are involved in the same way in the kernel function, as the $l2$ error is summed up.

However, this way is essentially problematic, since we want the machine to learn the location information.

Week 12

This week was not that productive since I was traveling in Washington DC. But I did have done some works and my code was running when I was away.

2 options

In regard of the 2 options I proposed last week, the option 2 is not valid in terms of the physics. Therefore, we are going to use only the option 1.

In terms of the option 1, however, we can also include the temperature to make an augmented u (like how we include `sf` and `cw` in the QBO1d experiments.) Note that, at each point $((lat, lon, t))$, the temperature has the shape exactly as the zonal wind u. The approach is to horizontally stack them.(this approach is mathematically well-posed).

subsampling

Subsampling is (may be) necessary for the training. In practice, we weight the samples according to $\cos(lat)$, which is proportional to the area of the grid box. (we call it uniform in space)

The motivation is that larger grid boxes contain more area-integrated momentum, so you want to focus on those grid boxes more if you want uniform performance "per unit of momentum".

Experiments

The main hyperparameters at this stage is the training samples. Here's the grid that I used:

- $\epsilon \in \{0.01, 0.001, 0.0001, 0.00001\}$
- $C \in \{16, 32, 128\}$
- `n_samples` $\in \{100, 1000, 10000, 20000, 40000, 60000\}$

Here's the result we get

1. `n_samples` $= 100$, $R^2 = 0.35$
2. `n_samples` $= 1000$, $R^2 = 0.45$
3. `n_samples` $= 10000$, $R^2 = 0.60$

We can see after `n_samples` $= 10000$, the improvement is fairly modest. In addition, the temperature inclusion way can give each setting 0.02 improvement in R^2

Week 13

This is (perhaps) the last week of the summer research.We started with some further discussion on the subsampling.

Subsampling revisited

Here's the question. Shall we use the space uniform data when testing? The choice means different mind set.

Subsampling can be viewed as 2 different stages in the pipeline.

1. subsampling creates a new dataset so we do the rest of the things with that dataset. In that case, the test dataset is also sampled by $\cos(lat)$.

2. subsampling is a strategy in training. Given the training dataset, we do subsampling so that the model can learn the more 'representative' information. In that case, the test dataset should not be sampled by $\cos(lat)$.

The ultimate concern is, when doing the online emulation, the distribution of the output is the same as the un-sampled one(the raw data). Therefore, use the un-sampled data to be the test data might be more suitable (since the offline-online correspondence is more rigorous).

In practice, we stick to the first way of using subsampling. It's true that when running online, the distribution of columns the GWP will see will not be weighted; however, the GWP's performance near the poles ought to be less important in determining whether it's able to produce a good mean state, or good QBO, and so on (because drags predicted for those cells have less effect on the global momentum budget). Thus, doing all training/testing with a weighted dataset is reasonable, if what you're after is good online performance.

Level-wise Offline Performance

The statistics we offered before is the average of all levels. Now we are curious about the loss(R^2) of each levels.

There are 40 levels in total. The results shows that the

- Offline performance from level 10-25 is quite good (R^2 around 0.9).
- Offline performance after level 25 suddenly becomes terrible (R^2 around 0.1).

Computational feasibility

Perhaps we can conclude is not the algorithm that can fulfill the requirement of computational feasibility.

First of all, in terms of the training. The required number for $R^2 = 0.6$ is around 10000. However, this already requires some time for training. Also, using 10000 numbers of vectors in online emulation run will lead to a relatively long execution time.

One way out is to use some trick to improve its efficiency from the very basic layer. For example, one way is to use a neural net to approximate the kernel function. Also, once we have completed this, we can retrain the model with the new kernel. This idea is very similar to pruning in neural network. The essence of an exponential function based kernel is to use the power of Taylor expansion to map the function to higher dimensional.

This idea seems quite promising, but there's one problem. If we adopt this method, why don't we directly use the neural network. With the neural network, we can do the training from end to end, instead of a 2-step training in the pipeline. The only good thing with support vector machine is the interpretability. But in terms of a regression problem, interpretability is not that important, at least not the difference maker.

References

[1] Baldwin, M. P. et al. The quasi-biennial oscillation. Reviews of Geophysics, 39(2), 179–229. https://doi.org/10.1029/1999rg000073

[2] A webpage of 1d QBO probelm. https://datawaveproject.github.io/qbo1d/html/model-description.html

[3] A Data-informed Framework for the Representation of Sub-grid Scale Gravity Waves to Improve Climate Prediction.

[4] Smola, Alex. J et al. A tutorial on support vector regression. Statistics and Computing 14: 199–222, 2004. https://alex.smola.org/papers/2004/SmoSch04.pdf

[5] Plumb, R. A. (1977). The interaction of two internal waves with the mean flow: Implications for the theory of the quasi-biennial oscillation. Journal of the Atmospheric Sciences, 34(12), 1847–1858. https://doi.org/10.1175/1520-0469(1977)034lt;1847:tiotiwgt;2.0.co;2

[6] Lindzen, R. S., amp; Holton, J. R. (1968). A theory of the quasi-biennial oscillation. Journal of the Atmospheric Sciences, 25(6), 1095–1107. https://doi.org/10.1175/1520-0469(1968)025lt;1095:atotqbgt;2.0.co;2

[7] Tipping, M. (1999). The relevance vector machine. Advances in neural information processing systems, 12. https://proceedings.neurips.cc/paper/1999/file/f3144cefe89a60d6a1afaf7859c5076b-Paper.pdf