Slide 1 Hi!

Hi, everyone. We would like to present to you the research paper published in 2021 by Takayuki Osogami Second order Techniques for Learning Time-series With Structural Breaks.

Slide 2 Challenges of time series forecasting

Initially, when we want to study time series we face some problems:

- Non-stationarity of data
- Structural breaks

A common approach to deal with these problems is to gradually forget past data replacing them with the new predicted results. However this approach has several downsides:
Hard to adapt and select the right constant forgetting rate (too high leads to poor adaptation to changes, too low results in overreaction to noise)
Appropriate regularisation is tricky (if size of data is too small, there is high risk of overfitting)

Slide 3 Novel Techniques

The paper proposes several techniques to overcome such obstacles:
Uses the “following best hyper forgetting rate”.
The paper proposes a technique for adaptively changing the regularisation strength and the forgetting rate in O(n^2) complexity in order to account for changes posed by structural breaks. Additionally, the suggested regularization is invariant to linear transformations.

Slide 4 Metamorphosis

Let’s consider a linear model (read from slide), where x is the vector of previous data points and thetas are their coefficients(weights). A standard approach in finding the thetas is using weighted mean squared error, where the gamma is the forgetting rate and is a hyperparameter. The optimal weights are used to predict the next y at t+1.

Slide 5 Proposed Solutions & Efficient Computations

In other words, the minimizer of WMSE is the inverse of Hessian of WMSE times negative gradient at the origin(gradient evaluated at x=0) and both of them are calculated through the following formulas.
As a matter of fact, we use a recursive formula (Sherman-Morrison lemma) for calculating the hessian for the next step with O(n^2) complexity.

Slide 6 Advancements in Regularization Techniques

The standard L2 regularisation is defined as following:
Unfortunately, it cannot be computed recursively, so the paper proposes another loss function:

The author proposes two lemmas that prove that the complexity of this algorithm for each iteration is O(n^2) and that regularisation is invariant to invertible linear transformations.
The algorithm uses the Sherman-Morrison lemma to find the next Hessian inverse if Hessian is invertible and pseudo-inverse algorithm if Hessian is not invertible. Unfortunately, the author forgot to mention how to check the matrix invertibility in n squared time.

Slide 7 Following best Hyper Forgetting Rate (больше текста)

We train several models with varied γ (forgetting rate) and λ (regularisation-coefficient).

Track Cumulative Squared Error (CSE) to evaluate model performance, where eta is the hyper forgetting rates which are recommended by the paper in the range from 0.9 to 1.

The eta with the minimum CSE is considered the best for the next time step (t+1).

Slide 8 Experiments

To empirically evaluate and test the proposed algorithm paper posed the following experiment questions:
How does the proposed regularisation compare against the L2 regularisation?
Can the proposed algorithm adaptively tune hyperparameters?
How does the proposed approach compare against existing methods for predicting non-stationary time series?

Slide 9 Experiment 1

Paper compared the predictions of the L2 regularisation and our model. Three orders are taken and the graphs show the change in root mean squared error with increasing regularization coefficient.
The algorithm occasionally outperforms L2 regularization, meaning the proposed model is reasonably effective.

Slide 10 Experiment 2: adaptiveness of the model

To check the adaptiveness of the model, the paper used the **synthetic 2000 point time series** with changepoints with a **structural break point at t = 1000.**

The time-series is learnt using the proposed algorithm and its predictive error is compared against the baseline with fixed gamma and lambda.

Slide 11 Comparison with other methods

Compared against ML methods for non-stationary time-series (vSGD, Almeida, HGD, Cogra, Adam, RMSProp and Adagrad)

Data is 10-years period of stock price indexes: S&P 500, Nikkei 225 and DAX.
The algorithm outperforms other ML

Slide 13

However, this outperformance could have been caused by minimization of the WMSE with the hyperparameters that minimize the CSE and the hyperparameters which minimize CSE. But the other baselines do not minimize this metric. Therefore no conclusion may be drawn - take a look at other matrices.
Furthermore, the proposed algorithm requires additional computational cost than baselines. (7.1 seconds for order 12 model against less than a second for the other models - add more specific time).

```
from scipy.linalg import LinAlgError
from sklearn.metrics import mean_squared_error

np.random.seed(0)

def Alg_1_exp1(df_spx, n, lambda_gg=-1, gamma_gg=-1, Nmod=30, Nhyp=11): # n = [1, 12]
    df = df_spx.copy()

    predicti, gamma_list, lambda_list = [], [], []

    # Step 1
    gamma = [0.01 + np.random.uniform(0.5 ** (1/n), 1) for i in range(Nmod)]
    lambda_i = [0] + [np.random.uniform(0, 1) for i in range(Nmod)]
    # print(f"Lambda = {lambda_i}")
    eta = [0.89 + 0.01 * j for j in range(Nhyp)] # ню
    # Step 2
    CSE = np.zeros(Nhyp)
    CSE_i = np.zeros((Nmod, Nhyp))
    # Step 3
    g = [np.zeros(n) for _ in range(Nmod)]
    H_inv = [np.zeros((n, n)) for _ in range(Nmod)]
    H = [np.zeros((n, n)) for _ in range(Nmod)]

    x_t = np.zeros(n)
    y_pred = np.zeros(Nmod)

    def prepare_feature_vector(df, t, x_t):
        previous_date = df.index[t - 1]
        x_t = np.delete(x_t, 0)
        x_t = np.append(x_t, df.Close.loc[previous_date])
```

```
        return np.array(x_t)

    def observe_target(df, t):
        current_date = df.index[t]
        # previous_date = df.index[t - 1]
        # y_t = abs((df.loc[current_date, 'Close'] - df.loc[previous_date, 'Close']) /
df.loc[previous_date, 'Close'])
        # # return current_date
        # print(y_t)
        return current_date
        # return y_t

    # def observe_target(df, t):
    #     current_date = df.index[t]
    #     previous_date = df.index[t - 1]
    #     y_t = abs((df.loc[current_date, 'Close'] - df.loc[previous_date, 'Close']) /
df.loc[previous_date, 'Close'])
    #     # return current_date
    #     return y_t

    def Update(H, x_flat, gamma):
        x = np.expand_dims(x_flat, 1)
        H_next_inv = gamma**-1 * H - ((gamma**-2 * H @ x @ x.transpose() @ H) /
                                (1 + gamma**-1 * x.transpose() @ H @ x))
        return H_next_inv

    for t in range(1, len(df)):
        ## Step 5
        x_t = prepare_feature_vector(df, t, x_t)

        ## Step 6 - find optimal CSE forgetting rate (eta)
        j_star = np.argmin(CSE)
        ## Step 7 - find the optimal model using the optimal CSE
        i_star = np.argmin(CSE_i[:, j_star])

        ## Step 8 - make prediction using optimal model
        y_pred[i_star] = x_t.T.dot(H_inv[i_star]).dot(g[i_star]) # r_t+1 - r_t = predict
        ## Step 9 - observe the real target
        y_t = observe_target(df, t)
        predicti.append(y_pred[i_star])
        # Step 10 - for each parameter value
        for j in range(Nhyp):
            # Step 11 - find the optimal model given the parameter
            i_star_j = np.argmin(CSE_i[:, j])
            #Step 12 - make prediction with the optimal model given the parameter
            CSE[j] += (y_pred[i_star_j] - y_t)**2

        for i in range(Nmod):
```

```
        ### Step 15 - for each model make a prediction
        y_pred[i] = x_t.T.dot(H_inv[i]).dot(g[i])

        ### Step 16 - for each parameter value
        for j in range(Nhyp):
            CSE_i[i,j] = eta[j] * CSE_i[i,j] + (y_pred[i] - y_t)**2
        if gamma_gg == -1:
            g[i] = g[i] * gamma[i] + x_t * (y_t)
        else:
            g[i] = g[i] * gamma_gg + x_t * (y_t)

        ### O(n**3)
        if gamma_gg == -1 and lambda_gg == -1:
            H[i] = gamma[i] * H[i] + np.outer(x_t, x_t) + lambda_i[i] * np.outer(x_t, x_t)
            if np.linalg.det(H[i]) > 0.000001:
                H_inv[i] = np.linalg.inv(H[i])
            else:
                H_inv[i] = np.linalg.pinv(H[i])
        elif lambda_gg == -1:
            H[i] = gamma_gg * H[i] + np.outer(x_t, x_t) + lambda_i[i] * np.outer(x_t, x_t)
            if np.linalg.det(H[i]) > 0.000001:
                H_inv[i] = np.linalg.inv(H[i])
            else:
                H_inv[i] = np.linalg.pinv(H[i])
        elif gamma_gg == -1:
            H[i] = gamma[i] * H[i] + np.outer(x_t, x_t) + lambda_gg * np.outer(x_t, x_t)
            if np.linalg.det(H[i]) > 0.000001:
                H_inv[i] = np.linalg.inv(H[i])
            else:
                H_inv[i] = np.linalg.pinv(H[i])
        else:
            H[i] = gamma_gg * H[i] + np.outer(x_t, x_t) + lambda_gg * np.outer(x_t, x_t)
            if np.linalg.det(H[i]) > 0.000001:
                H_inv[i] = np.linalg.inv(H[i])
            else:
                H_inv[i] = np.linalg.pinv(H[i])
    if gamma_gg == -1:
        gamma_list.append(gamma[np.argmin(CSE)])
    else:
        gamma_list.append(gamma_gg)
    if lambda_gg == -1:
        lambda_list.append(lambda_i[np.argmin(CSE)])
    else:
        lambda_list.append(lambda_gg)
df['gamma'] = [0] + gamma_list
df['lambda'] = [0] + lambda_list
df['pred_alg1'] = [0] + predicti
return df
```

df_oil = df_oil.rename(columns={"Price": "Close"})
data_to_predict = pd.dataFrame()
data_to_predict['Close'] = df_oil['Close'][:-13]
df = Alg_1_exp1(data_to_predict, 1,)
[redacted]
ChatGPT
It seems there are several issues in the code provided. Let's go through them step by step and make the necessary corrections:

Import required libraries:

python

Copy code

```
import        as     import          as
```

Define the DataFrame   before using it in the function.
There is a typo in the line   . It should be
                  .

Here's the corrected version of the code:

python

Copy code

```
import        as     import          as     from                  import             from
                import                                            0
                              def Alg_1_exp1                   1             1
     30       11

                'Close'             'Close'     13
1
```

Make sure to replace   with your own DataFrame or load the DataFrame properly before calling the   function.

from scipy.linalg import LinAlgError
from sklearn.metrics import mean_squared_error

```
np.random.seed(0)

def Alg_1_exp1(df_spx, n, lambda_gg=-1, gamma_gg=-1, Nmod=30, Nhyp=11): # n = [1, 12]
    df = df_spx.copy()

    predicti, gamma_list, lambda_list = [], [], []

    # Step 1
    gamma = [0.01 + np.random.uniform(0.5 ** (1/n), 1) for i in range(Nmod)]
    lambda_i = [0] + [np.random.uniform(0, 1) for i in range(Nmod)]
    # print(f"Lambda = {lambda_i}")
    eta = [0.89 + 0.01 * j for j in range(Nhyp)] # ню
    # Step 2
    CSE = np.zeros(Nhyp)
    CSE_i = np.zeros((Nmod, Nhyp))
    # Step 3
    g = [np.zeros(n) for _ in range(Nmod)]
    H_inv = [np.zeros((n, n)) for _ in range(Nmod)]
    H = [np.zeros((n, n)) for _ in range(Nmod)]

    x_t = np.zeros(n)
    y_pred = np.zeros(Nmod)

    def prepare_feature_vector(df, t, x_t):
        previous_date = df.index[t - 1]
        x_t = np.delete(x_t, 0)
        x_t = np.append(x_t, df.Close.loc[previous_date])
        return np.array(x_t)

    def observe_target(df, t):
        current_date = df.index[t]
        # previous_date = df.index[t - 1]
        # y_t = abs((df.loc[current_date, 'Close'] - df.loc[previous_date, 'Close']) /
df.loc[previous_date, 'Close'])
        # # return current_date
        # print(y_t)
        return current_date
        # return y_t

    # def observe_target(df, t):
    #     current_date = df.index[t]
    #     previous_date = df.index[t - 1]
    #     y_t = abs((df.loc[current_date, 'Close'] - df.loc[previous_date, 'Close']) /
df.loc[previous_date, 'Close'])
    #     # return current_date
    #     return y_t

    def Update(H, x_flat, gamma):
```

```
    x = np.expand_dims(x_flat, 1)
    H_next_inv = gamma**-1 * H - ((gamma**-2 * H @ x @ x.transpose() @ H) /
                        (1 + gamma**-1 * x.transpose() @ H @ x))
    return H_next_inv

for t in range(1, len(df)):
    ## Step 5
    x_t = prepare_feature_vector(df, t, x_t)

    ## Step 6 - find optimal CSE forgetting rate (eta)
    j_star = np.argmin(CSE)
    ## Step 7 - find the optimal model using the optimal CSE
    i_star = np.argmin(CSE_i[:, j_star])

    ## Step 8 - make prediction using optimal model
    y_pred[i_star] = x_t.T.dot(H_inv[i_star]).dot(g[i_star]) # r_t+1 - r_t = predict
    ## Step 9 - observe the real target
    y_t = observe_target(df, t)
    predicti.append(y_pred[i_star])
    # Step 10 - for each parameter value
    for j in range(Nhyp):
        # Step 11 - find the optimal model given the parameter
        i_star_j = np.argmin(CSE_i[:, j])
        #Step 12 - make prediction with the optimal model given the parameter
        CSE[j] += (y_pred[i_star_j] - y_t)**2

    for i in range(Nmod):
        ### Step 15 - for each model make a prediction
        y_pred[i] = x_t.T.dot(H_inv[i]).dot(g[i])

        ### Step 16 - for each parameter value
        for j in range(Nhyp):
            CSE_i[i,j] = eta[j] * CSE_i[i,j] + (y_pred[i] - y_t)**2
        if gamma_gg == -1:
            g[i] = g[i] * gamma[i] + x_t * (y_t)
        else:
            g[i] = g[i] * gamma_gg + x_t * (y_t)

        ### O(n**3)
        if gamma_gg == -1 and lambda_gg == -1:
            H[i] = gamma[i] * H[i] + np.outer(x_t, x_t) + lambda_i[i] * np.outer(x_t, x_t)
            if np.linalg.det(H[i]) > 0.000001:
                H_inv[i] = np.linalg.inv(H[i])
            else:
                H_inv[i] = np.linalg.pinv(H[i])
        elif lambda_gg == -1:
            H[i] = gamma_gg * H[i] + np.outer(x_t, x_t) + lambda_i[i] * np.outer(x_t, x_t)
            if np.linalg.det(H[i]) > 0.000001:
```

```
                    H_inv[i] = np.linalg.inv(H[i])
                else:
                    H_inv[i] = np.linalg.pinv(H[i])
            elif gamma_gg == -1:
                H[i] = gamma[i] * H[i] + np.outer(x_t, x_t) + lambda_gg * np.outer(x_t, x_t)
                if np.linalg.det(H[i]) > 0.000001:
                    H_inv[i] = np.linalg.inv(H[i])
                else:
                    H_inv[i] = np.linalg.pinv(H[i])
            else:
                H[i] = gamma_gg * H[i] + np.outer(x_t, x_t) + lambda_gg * np.outer(x_t, x_t)
                if np.linalg.det(H[i]) > 0.000001:
                    H_inv[i] = np.linalg.inv(H[i])
                else:
                    H_inv[i] = np.linalg.pinv(H[i])
        if gamma_gg == -1:
            gamma_list.append(gamma[np.argmin(CSE)])
        else:
            gamma_list.append(gamma_gg)
        if lambda_gg == -1:
            lambda_list.append(lambda_i[np.argmin(CSE)])
        else:
            lambda_list.append(lambda_gg)
    df['gamma'] = [0] + gamma_list
    df['lambda'] = [0] + lambda_list
    df['pred_alg1'] = [0] + predicti
    return df


data_to_predict = pd.DataFrame()
data_to_predict['Close'] = df_oil['Close'][:-13]
df = Alg_1_exp1(data_to_predict, 1)
# df.gamma.unique
```

```
<ipython-input-51-b39ecb69b32a> in <cell line: 129>()
    127 data_to_predict = pd.DataFrame()
    128 data_to_predict['Close'] = df_oil['Close'][:-13]
--> 129 df = Alg_1_exp1(data_to_predict, 1)
    130 # df.gamma.unique

<ipython-input-51-b39ecb69b32a> in Alg_1_exp1(df_spx, n, lambda_gg, gamma_gg, Nmod, Nhyp)
     72             i_star_j = np.argmin(CSE_i[:, j])
     73             #Step 12 - make prediction with the optimal model given the parameter
---> 74             CSE[j] += (y_pred[i_star_j] - y_t)**2
     75 
     76         for i in range(Nmod):
```

TypeError: unsupported operand type(s) for -: 'numpy.float64' and 'Timestamp'

ChatGPT

The error is occurring because you are trying to perform a mathematical operation between a and a object, which is not supported. It seems like is of type when it should be a numerical value.

In the function , you are returning the variable , which seems to be a timestamp. You may need to modify the function to return the expected target value, which should be numerical.

Here's the modified version of the function:

python

Copy code

```
def observe_target
1 abs 'Close' 'Close'
'Close' return
```

Replace the existing function with the one provided above and rerun the code. This will ensure that is a numerical value, and the mathematical operation will work as expected.

Slide 12

In experiment 3 we apply the algorithm to financial time-series and compare its predictive accuracy against other machine learning methods. We thus compare algorithm 1 against other baseline optimizers such as vSGD. Almeida, HGD and ADAM. The results are summarised in the graph and it shows that algorithm 1 outperforms the baselines. The potential reason for this is that it adaptively tunes the hyperparameters of the model, while baselines do not. Nonetheless we can still conclude that the proposed algorithm may be suitable for making predictions.
Slide 13 Conclusion
Overall the results of the experiments show that the proposed regularisation outperforms L2. Algorithm 1 can quickly tune hyperparameters following a structural break and it outperforms baselines.

Slide 13 - Conclusion

The article derives three main conclusions regarding the proposed method.

Firstly, the results clearly demonstrate that the proposed regularisation method may under some situations be better than L2. Secondly, Algorithm 1 is able to swiftly adjust hyperparameters following a structural break. Lastly, the proposed method outperforms baseline predictive solutions.

Extra Outcome Slide:

For the extra outcome segment of our project, we decide to compare the proposed algorithm with state-of-the-art models, evaluating its performance in predicting the price of oil. We hose this dataset because of the high occurrence of structural breaks, due to geopolitical and economic factors.

The results, displayed in the table, illustrate that the proposed algorithm performs comparably to an ARIMA model. However, the ARIMA model requires extensive recalibration after each prediction to adapt to potential structural breaks, making it significantly slower than our proposed algorithm. This computational cost may be of the essence in high-frequency trading environments where minimal time lags may lead to substantial profits or losses. This suggests the potential area of application for the proposed methodology. With this I would like to conclude our presentation and now we will proceed to a demonstrate of our code.