KDDE

Deep Learning Methods for Predictive Tasks with Large Scale Sensor Data

Roberto Corizzo

G2NET
COST ACTION CA17137
A NETWORK FOR GRAVITATIONAL WAVES, GEOPHYSICS AND MACHINE LEARNING

1st Conference on Machine Learning for Gravitational Waves, Geophysics, Robotics and Control System

About me

Roberto Corizzo

Postdoc research associate

Department of Computer Science

University of Bari Aldo Moro, Italy

Research Interests

Data Mining and Knowledge Discovery

Big Data Analytics

Predictive Models for Sensor Data

Knowledge Discovery and Data Engineering research group
http://kdde.di.uniba.it/

Research focus

- Development of predictive models, where data are continuously produced at regular time intervals by sensors placed on geo-referenced nodes.

- The goal is to predict the values assumed by a target feature of interest in the following time instants (few minutes to few days ahead).

Issues and challenges

- Spatial and temporal autocorrelation introduced by spatial proximity of nodes and the cyclical nature of the days

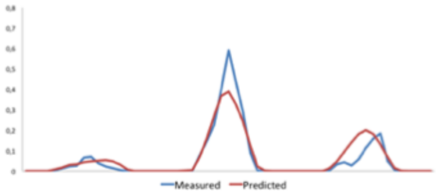
- Noisy data (outliers / missing values due to fault on sensors)

- Abundance of large-scale data

Applications with Sensor Data

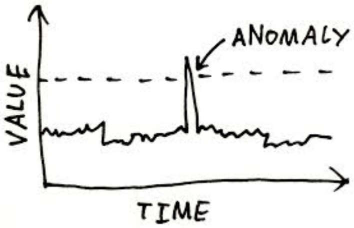
Predictive modeling of renewable energy production

- For a network of photovoltaic (PV) plants spread over a defined geographical area and connected to a power grid
- Exploiting historical data and real time data of production, continuously produced at regular time intervals by sensors placed on each plant of interest
- Exploiting weather and irradiance predictions



Online anomaly detection and repair for sensor data in energy plants

- To guarantee accurate predictions in presence of noisy or missing data



Practical importance in smart grids

- Grid integration
- Load balancing
- Energy trading

Self-Organizing Maps: overview, possible tasks and applications

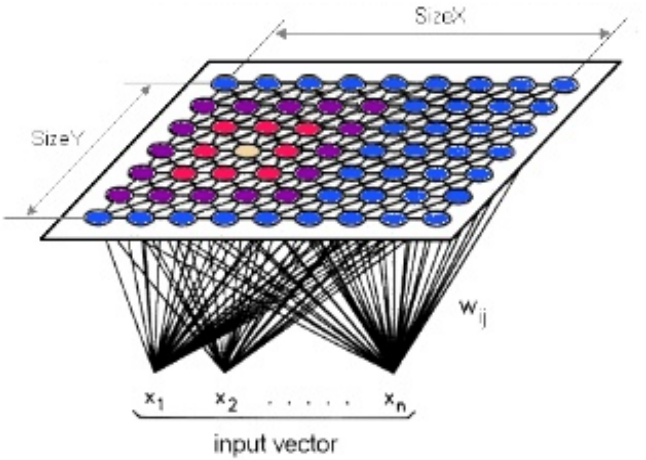
Self-Organizing Map (SOM)

Popular algorithm for unsupervised analysis of data, exploited for a variety of tasks such as exploratory analysis, financial diagnosis, fraud detection, etc.

Maps a high-dimensional input data onto a 2D (or 3D) grid (feature map) of neurons.

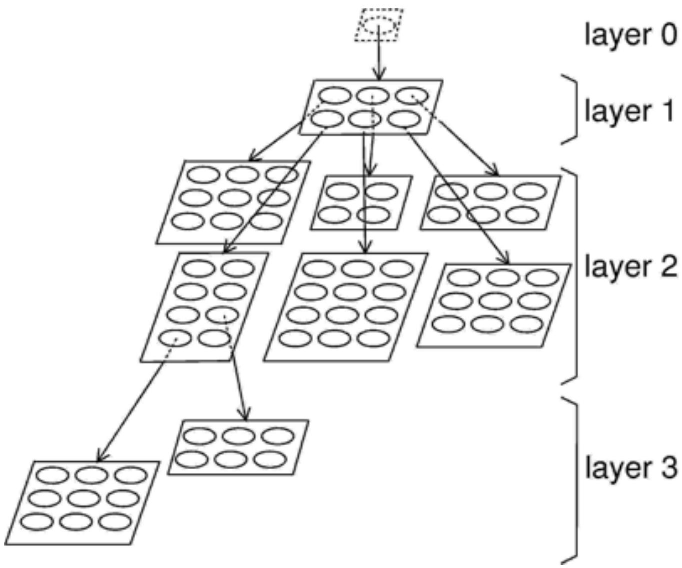
Limitations

- Static architecture
- Shape and number of neurons in the feature map needs to be determined before the training.



Growing Hierarchical Self-Organizing Map (GHSOM)

Multi-level tree-like architecture consisting of individual SOMs



Self-Organizing Maps Recent research works

- **Hsu (2006)** extended SOM algorithm for categorical data;
- **Ippoliti et al. (2012)** presented an online method for network anomaly detection exploiting GHSOM models;
- **Huang et al. (2012)** introduced a predictive approach for the binary classification setting tested on KDD CUP 1999 dataset;
- **Quintian et al. (2014)** proposed a hybrid regression system for solar energy prediction in which SOM models are used for clustering, subsequently exploited by local models;
- **Sarazin et al. (2014)** proposed a distributed biclustering algorithm for Apache Spark based on the SOM model.
- **Zurita et al. (2018)** exploit SOM to model the operating conditions of an industrial process reflected by available auxiliary time series.

GHSOM Algorithm

Parameters: *epochs*, τ_1, τ_2

- Level-0 neuron: mqe_0 is calculated with respect to all the input instances
- First neuron map m_1 is created at Level-1 consisting of 2×2 neurons
- m_1 trained using the conventional SOM training process (competitive learning)

τ_1 controls the growth process

τ_2 controls the minimum granularity of data expected to be represented by each neuron.

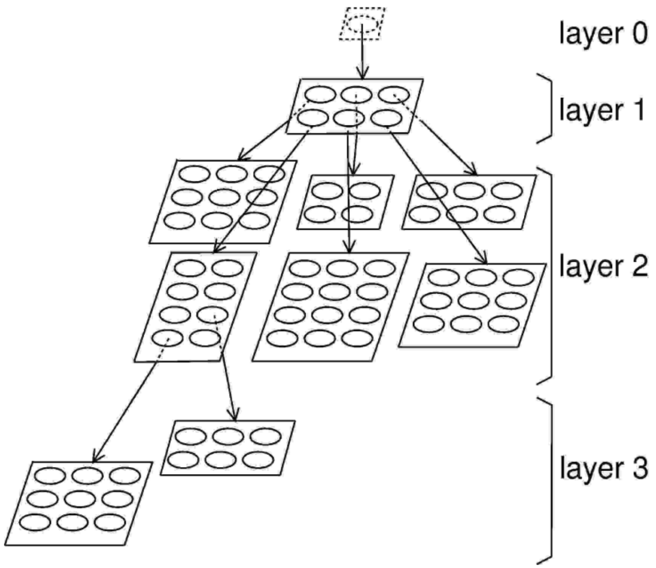
Mean Quantization Error (MQE) of a neuron is the total deviation of the neuron from its mapped input instances.

GHSOM Algorithm

Training process

For a specified number of *epochs*:

- For each instance from training data, the neuron with the minimum distance from each input instance is selected as the winner neuron *c*
- *c* and its surrounding neighbour neurons are adapted towards the input instance.



Neuron adaptation

- $h_{ck}(t)$ is the neighbourhood factor for a neuron m_k with respect to the winner neuron *c* for input instance $x(t)$ presented at time *t*.

$$m_k(t_e) = \frac{\sum_{t=t_s}^{t_e} h_{ck}(t) * x(t)}{\sum_{t=t_s}^{t_e} h_{ck}(t)}$$

- Map *m* is analysed and MQE_m is computed. *m* grows until the following criterion is satisfied:

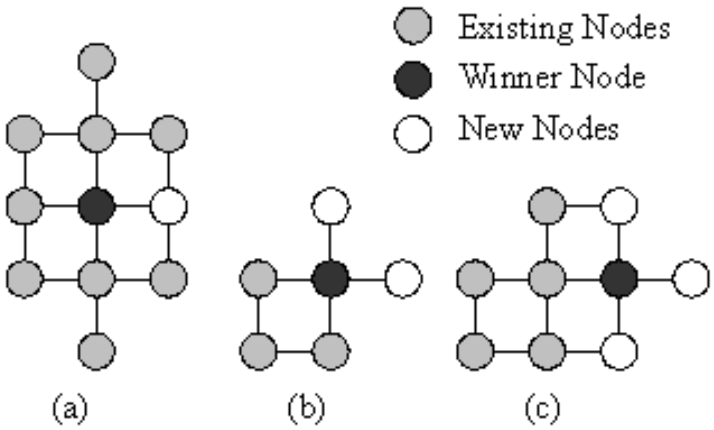
$$MQE_m < \tau_1 \cdot mqe_p$$

MQE of the parent neuron *p* from which this map *m* is expanded

GHSOM Algorithm

Growing process

- The neuron with the highest MQE is identified as the error neuron *e*.
- Its most dissimilar direct neighbouring neuron *d* is selected and a new row (or column) is inserted between *e* and *d*.
- Vectors of new neurons are initialized as the average of the weight vectors of their adjacent neighbours.
- The grown layer is trained and analysed again.



GHSOM Algorithm

Hierarchical growth process

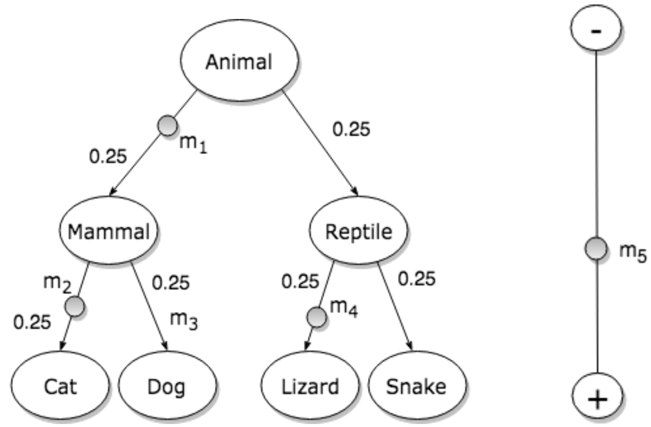
- Once the $\boldsymbol{\tau_1}$ criterion is satisfied, each neuron in the map is analysed according to this criterion:

$$mqe_k < \boldsymbol{\tau_2} \cdot mqe_0$$

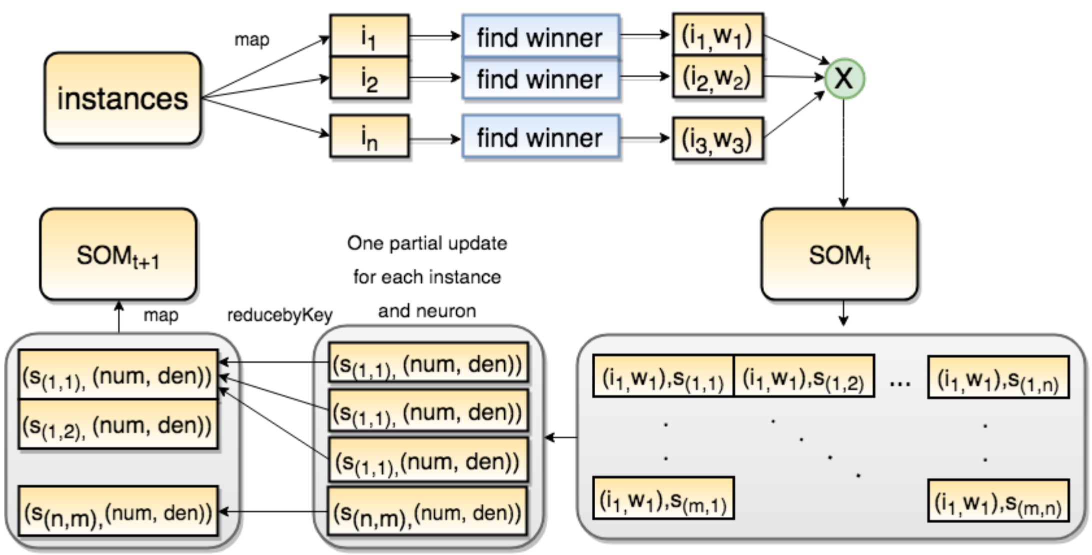
- The neurons which do not satisfy the $\boldsymbol{\tau_2}$ criterion are expanded into new maps at the next level of hierarchy (same process of training, growth and hierarchical expansion as the level-1 map).
- The training of the GHSOM stops when all the neurons in the lowest level maps satisfy the $\boldsymbol{\tau_2}$ criterion.
- The resulting GHSOM structure thus contains multiple SOM layers arranged in a hierarchy with each SOM representing the data at a finer granularity than its parent layer.

Spark-GHSOM: Contributions

- Distance hierarchy approach to modify the optimization function of GHSOM so that it can (also) coherently handle mixed-attribute datasets.



- Distributed implementation of GHSOM in Apache Spark: it formulates the training process, including the two-dimensional growth and the hierarchical growth process adopting *map* and *reduceByKey* functions.



Spark-GHSOM: Predictive Module

Training time

- Keep track of the target value of labeled instances in the respective winner neuron
- Assign one definitive target value to each neuron, calculated as the average of all assignments received during the training process

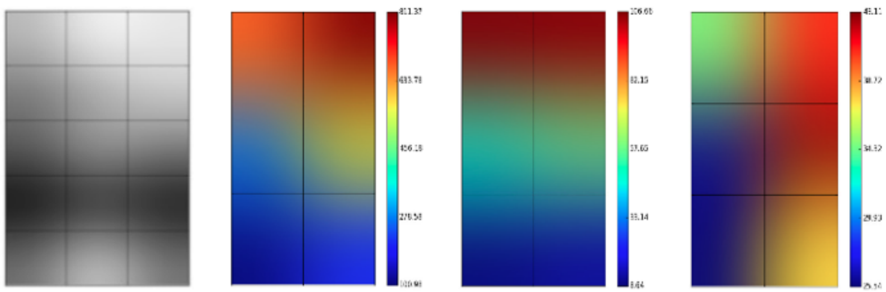
Testing time

- Given an instance $\boldsymbol{x}$, for each SOM, find the closest neuron $\boldsymbol{c}$ with a target attribute value $\boldsymbol{c_y}$ assigned, and add it to a candidate list $\boldsymbol{C}$
- Choose the closest neuron $\boldsymbol{c}$ in $\boldsymbol{C}$ w.r.t $\boldsymbol{x}$ in terms of Euclidean distance
- Assign its target value: $\boldsymbol{x_y = c_y}$

Spark-GHSOM: Experimental Results

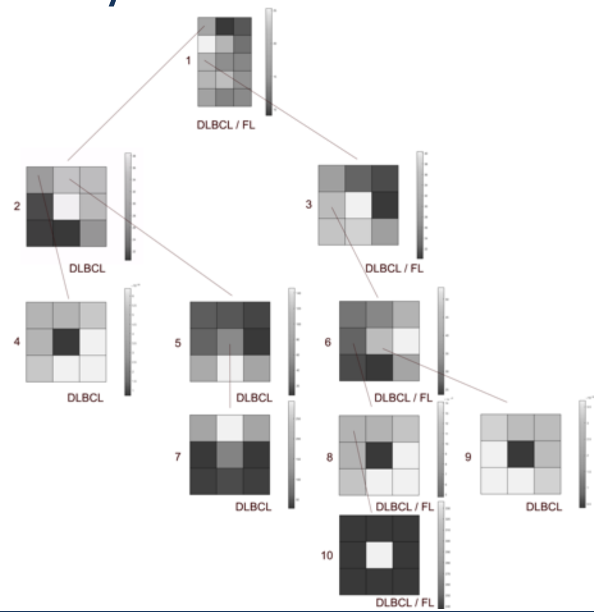
Qualitative results

Census data



(a) U-Matrix (b) cap.gains (c) wage/hr (d) age

Microarray data



Quantitative results

Sensor data forecasting

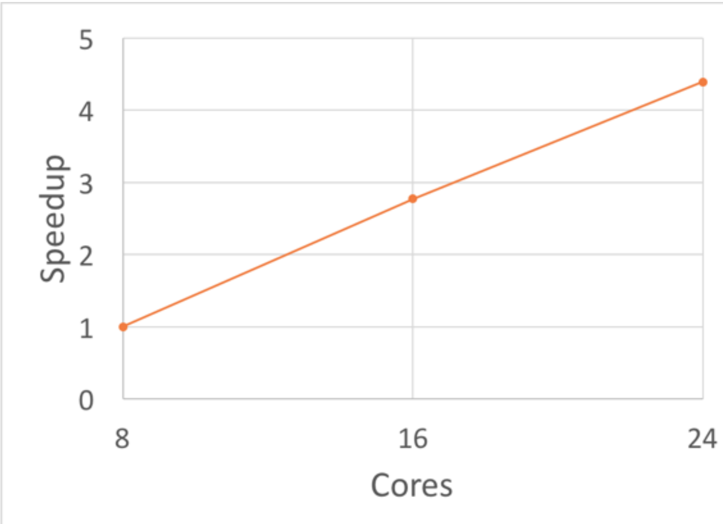
Dataset:	PV NREL		PV Italy		Burlington	
Target:	power		power		power	
Method	**RMSE**	**Impr.%**	**RMSE**	**Impr.%**	**RMSE**	**Impr.%**
ARIMA	0.2849		0.1675		0.2263	
K-Means	0.2336	17.99	0.1507	10.00	0.1397	38.26
SVR (Linear)	0.1781	37.49	0.1967	-17.43	0.1953	13.70
SVR (Poly)	**0.1491**	47.67	0.1758	-4.96	0.1623	28.28
SVR (Sigmoid)	>1		0.1966	-17.37	0.1952	13.74
Isotonic Reg.	0.2621	8.00	0.2000	-19.40	0.4388	-93.90
Linear Reg.	0.2307	19.02	0.1503	10.27	0.1451	35.88
Spark-GHSOM	0.2309	18.92	**0.1340**	**19.94**	**0.1370**	**39.45**

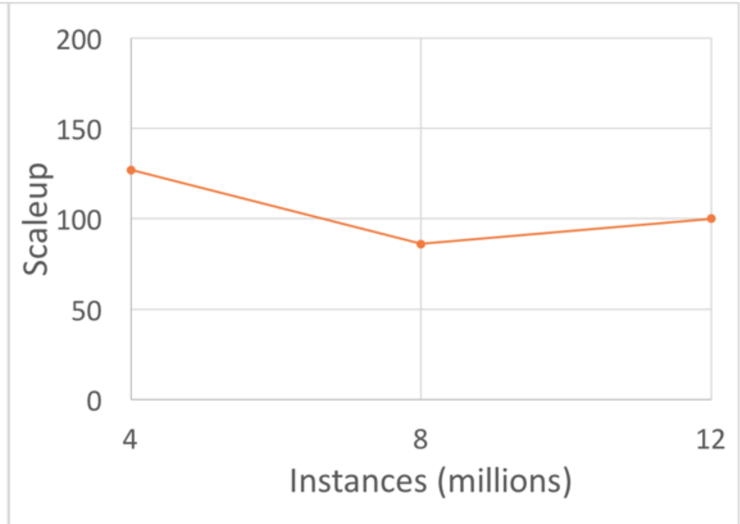
Regression with mixed attributes

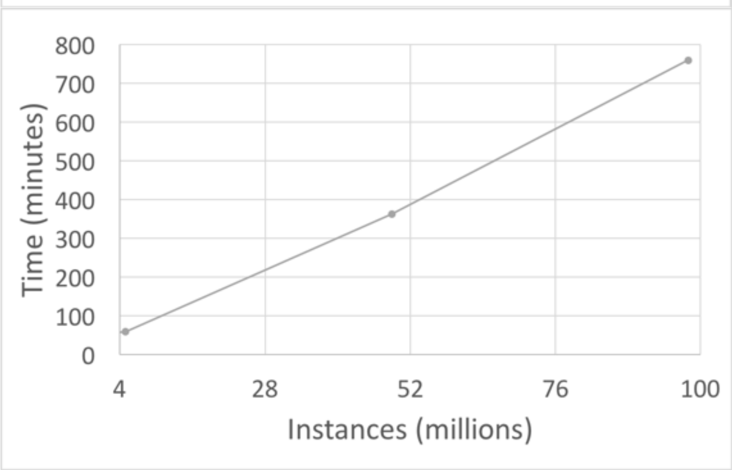
Pairwise comparison	*p*-value	winner
RMSE criterion		
Spark-GHSOM VS		
SVR (Linear)	**0.011**	**Spark-GHSOM**
SVR (Poly)	**0.038**	**Spark-GHSOM**
SVR (Sigmoid)	**0.011**	**Spark-GHSOM**
Linear Reg.	**0.036**	**Spark-GHSOM**
Isotonic Reg.	**0.008**	**Spark-GHSOM**

Spark-GHSOM: Experimental Results

Scalability results







KDD CUP 1999 dataset. Stress test with a fixed configuration of the parameters τ1= 0.5, τ2= 0.5 and epochs= 15.

Auto-Encoders: overview, possible tasks and applications

Auto-Encoders

Auto-Encoders learn to reconstruct a given input representation with a low reconstruction error.

A suitable way to learn an auto-encoder consists in layer-wise back-propagation learning.

Each auto-encoder has an encoding function **γ** and a decoding function **δ** such that:

$$\gamma : \mathcal{X} \rightarrow \mathcal{F}, \qquad \delta : \mathcal{F} \rightarrow \mathcal{X}$$

$$\gamma, \delta = \arg\min_{\gamma,\delta} \|X - \delta(\gamma(X))\|^2$$

Auto-Encoders

Encoding stage (one hidden layer)

Takes the input $\boldsymbol{x} \in \mathbf{R}^d = \mathbf{X}$ and maps it to an hidden representation $\boldsymbol{z} \in \mathbf{R}^p = \mathbf{F}$

$$\mathbf{z} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

Where **σ** is a sigmoid or a rectified linear unit activation function, **W** is a weight matrix and **b** is a bias vector.

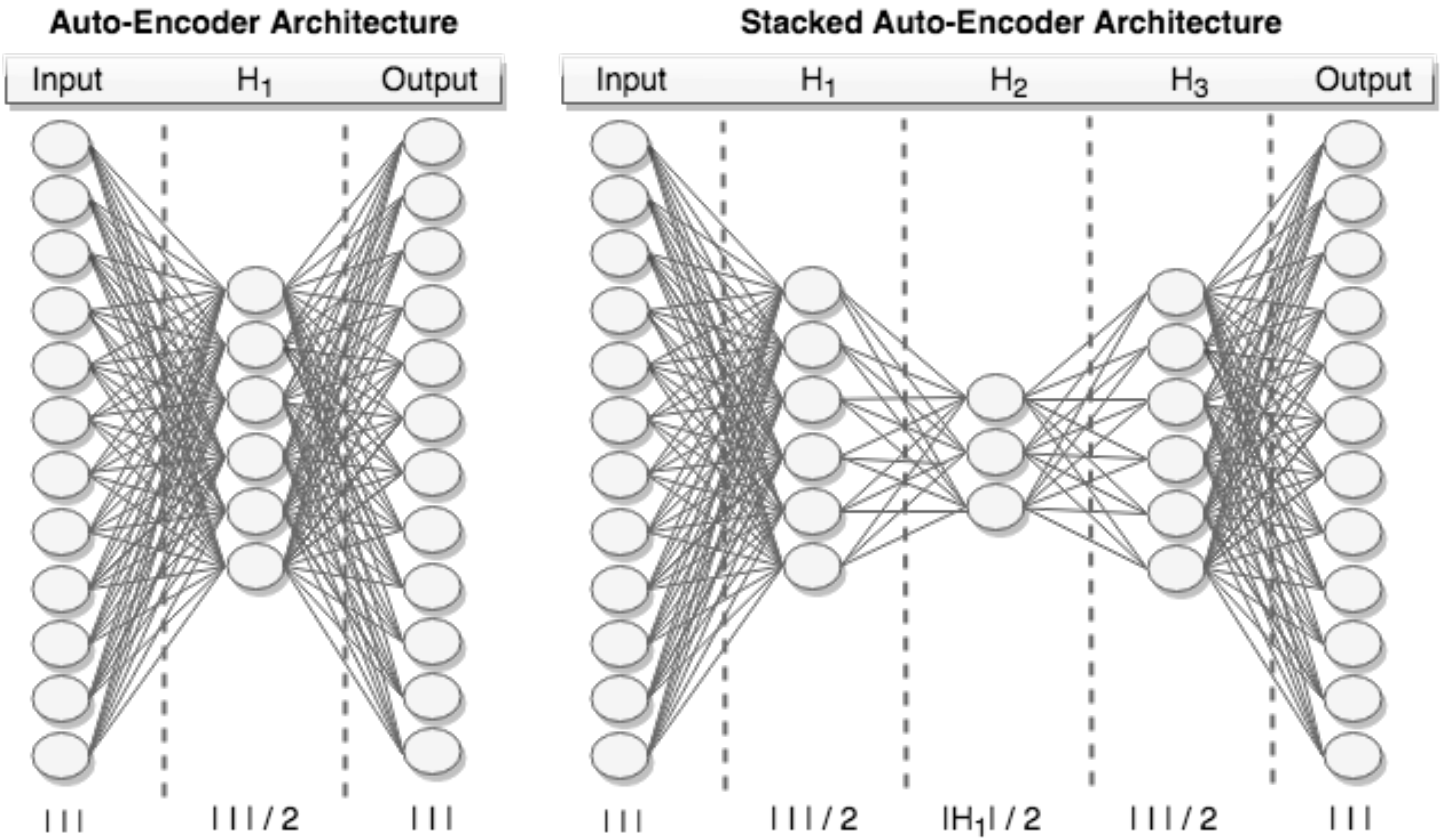
Decoding stage (one hidden layer)

The decoding stage reconstructs *x* from *z* as: $\mathbf{x}' = \sigma'(\mathbf{W}'\mathbf{z} + \mathbf{b})$

such that the following loss is minimized:

$$\mathcal{L}(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|^2 = \|\mathbf{x} - \sigma'(\mathbf{W}'(\sigma(\mathbf{W}\mathbf{x} + \mathbf{b})) + \mathbf{b}')\|^2$$

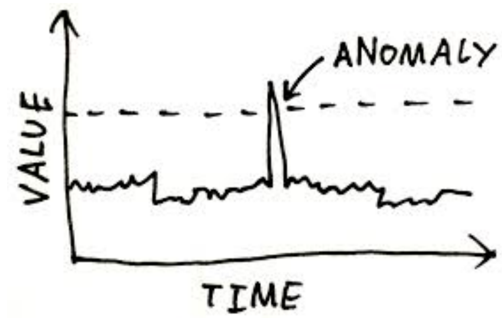
Auto-Encoders vs Stacked Auto-Encoders



Auto-Encoders: Possible tasks

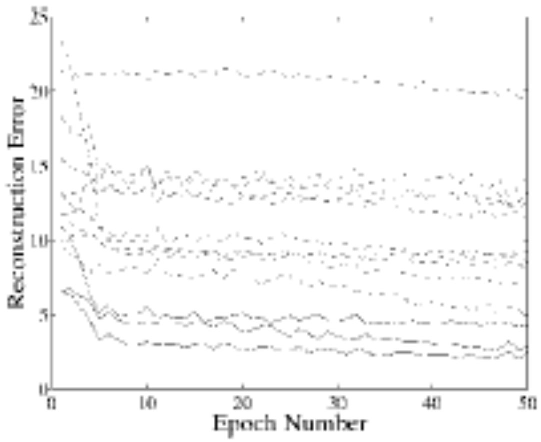
Anomaly detection

- Once the auto-encoder is trained with non-anomalous data, a high reconstruction error for a new instance means that it is possibly an anomaly.



Clustering

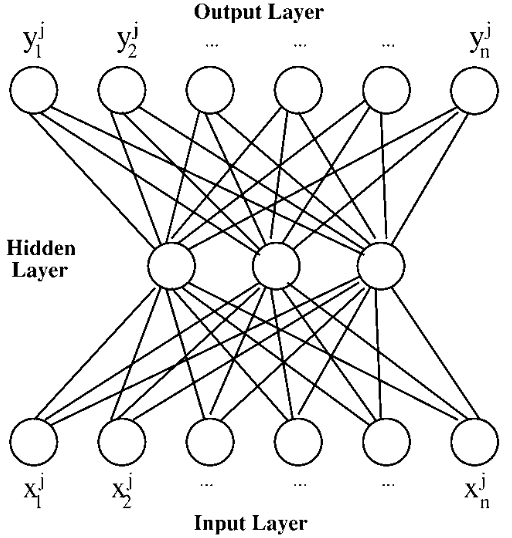
- Non-linear auto-encoders build multiple-local-valley representations of the underlying domain.
- Instances with similar values of reconstruction error may imply that they belong to the same cluster.



Auto-Encoders: Possible tasks

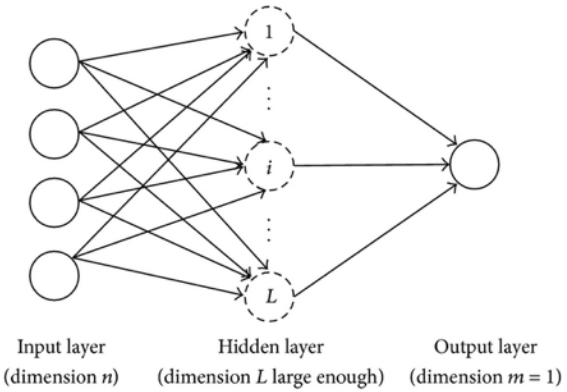
Recognition-based classification

- Once trained with data belonging to the positive class, if the **reconstruction error** is lower than a **threshold** for an unseen example, it belongs to the positive class, otherwise it belongs to the negative class.



Concept learning prior to classification or regression

- Perform **layer-wise pre-training**
- Trained layers can be copied to other neural network models (a new model with one output neuron for classification)
- Pre-training should initialize the weights closer to good solutions (see Larochelle et al. 2009)

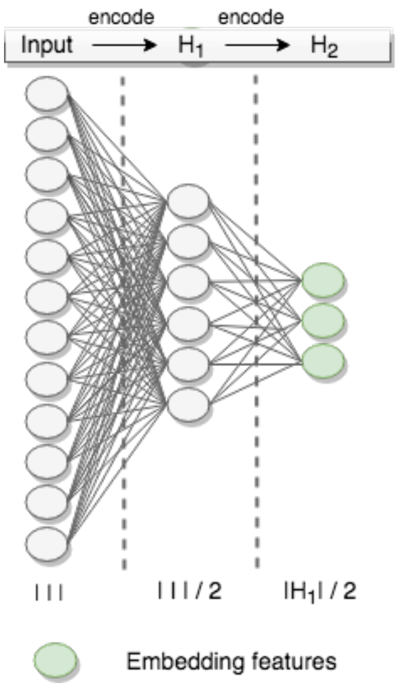


Auto-Encoders: Possible tasks

Feature extraction

- After training, extract a set of features of reduced dimensionality (**embedding features**) exploiting the encoding function.

- Reduced dimensionality implies model compactness and possible mitigation of collinearity effects, similarly to Principal Component Analysis (PCA).



Note: Auto-encoder embedding features are equivalent to PCA just if the hidden layer has linear activations (see Japkowicz et al. 2000)

Stacked Auto-Encoders in Apache Spark

Code example in Scala

```scala
import org.apache.spark.ml.scaladl.{MultilayerPerceptronClassifier, StackedAutoencoder}
val train = spark.read.format("libsvm").option("numFeatures", 784).load(mnistTrain).persist()
train.count()
val stackedAutoencoder = new StackedAutoencoder().setLayers(Array(784, 32))
  .setInputCol("features")
  .setOutputCol("output")
  .setDataIn01Interval(true)
  .setBuildDecoder(false)
val saModel = stackedAutoencoder.fit(train)
val autoWeights = saModel.encoderWeights
val trainer = new MultilayerPerceptronClassifier().setLayers(Array(784, 32, 10)).setMaxIter(1)
val initialWeights = trainer.fit(train).weights
System.arraycopy(autoWeights.toArray, 0, initialWeights.toArray, 0, autoWeights.toArray.length)
trainer.setInitialWeights(initialWeights).setMaxIter(10)
val model = trainer.fit(train)
```

https://github.com/avulanov/scalable-deeplearning

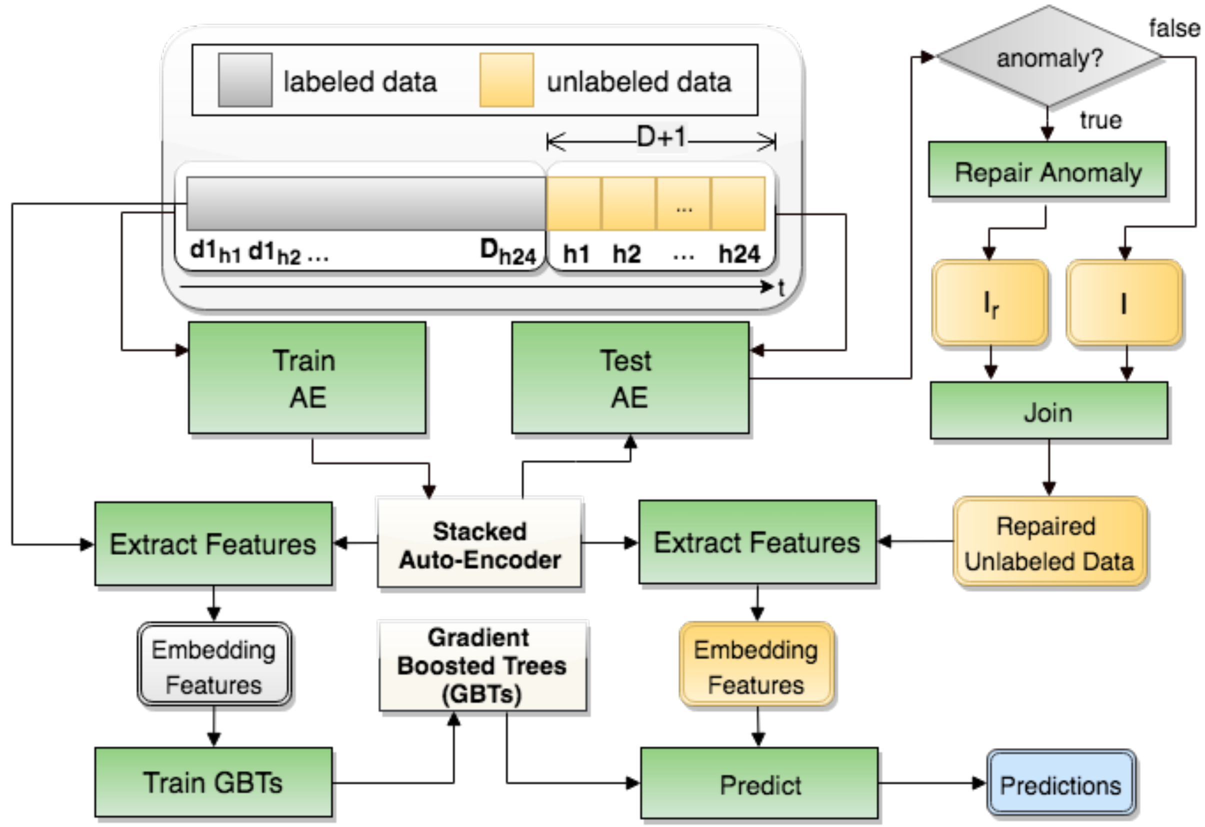
Anomaly Detection and Repair for Accurate Predictions in Geo-distributed Big Data

Research carried out during a visiting period at American University in Washington D.C under the supervision of Prof. Nathalie Japkowicz.

Anomaly Detection and Repair for Accurate Predictions in Geo-distributed Big Data

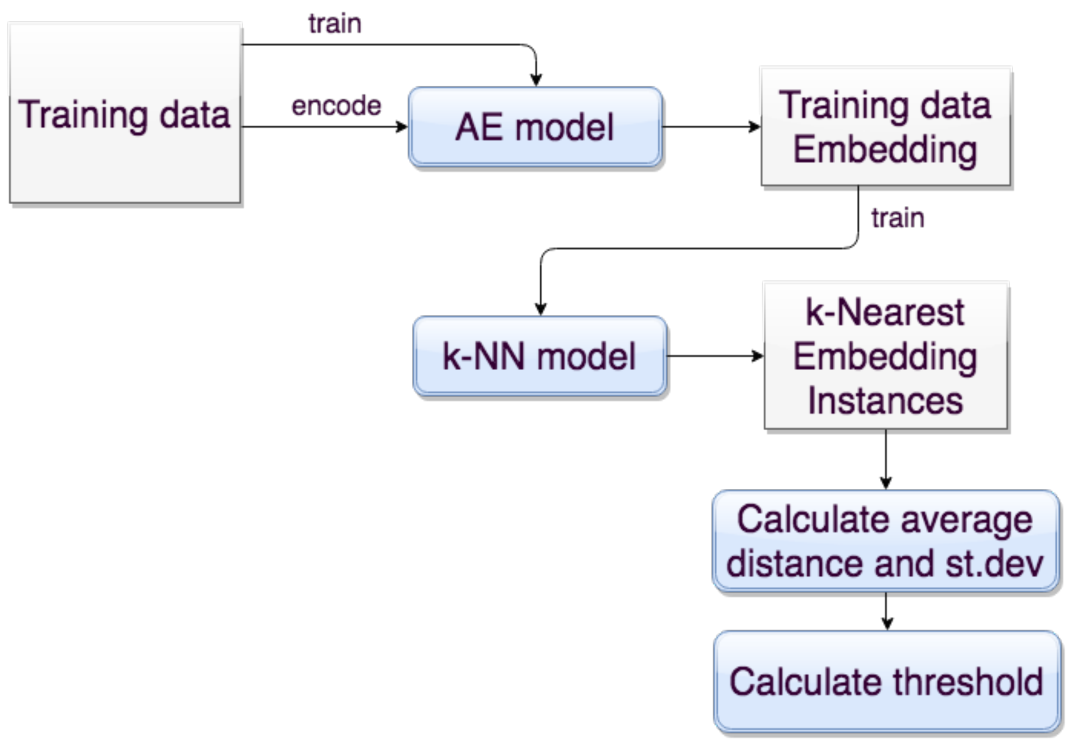
KDDE

- Auto-encoder based anomaly detection
- Non-selective and selective data repair exploiting normal instances from other sites, using a closeness factor, dependent from the spatial distance between sites locations (in *km*).



Anomaly Detection

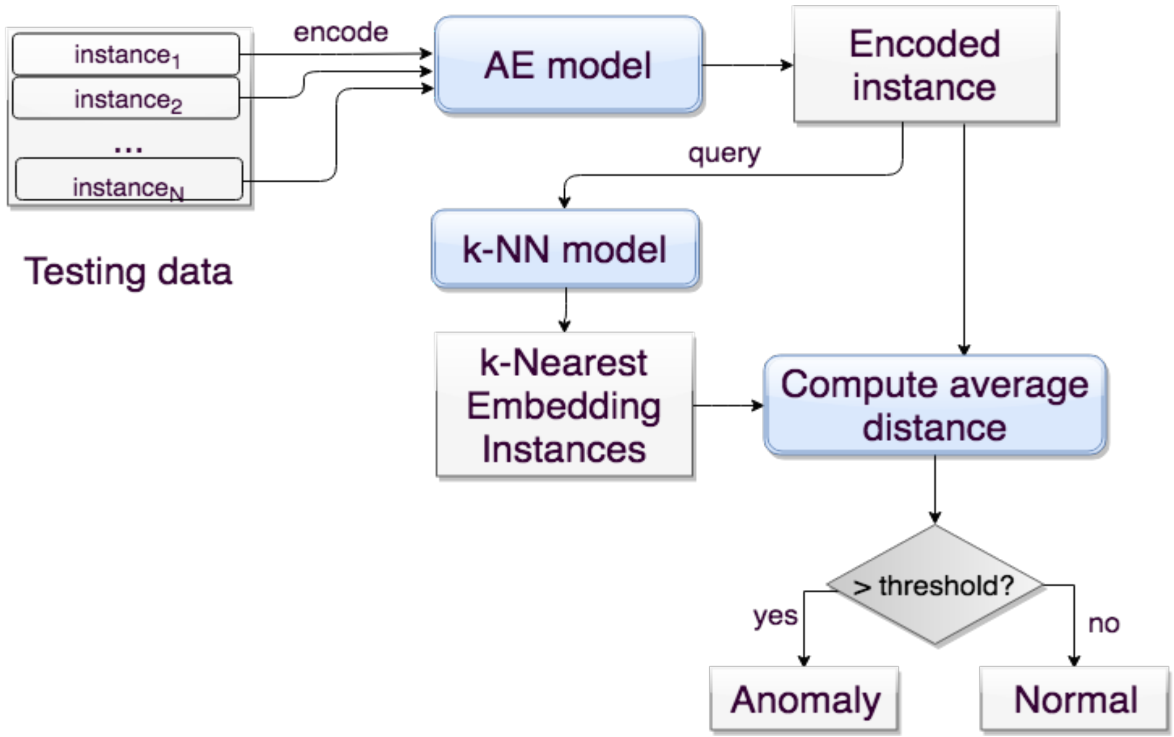
k-NN based



Training phase

Anomaly Detection

k-NN based



Detection phase

Data Repair

Non-selective: Entire instance ***x*** (all features) repaired exploiting non-anomalous instances of other sites by a weighted average.

$$\mathbf{x}'_{(p,t)} = \frac{\sum_{p' \in N(p)} \left[\mathbf{x}_{(p',t)} \cdot \left(1 - \frac{dist(p,p')}{\max Dist(P)}\right) \right]}{\sum_{p' \in N(p)} \left(1 - \frac{dist(p,p')}{\max Dist(P)}\right)}$$

Selective: For ***each feature*** of an anomalous instance ***x***, it is detected whether the observed value is abnormal, querying for each site its ***historical data*** at the same hour of the same month.

$$\mathbf{x}'_{(p,t)}[v] \leftarrow \frac{\sum_{p' \in N(p)} \left[\mathbf{x}_{(p',t)}[v] \cdot \left(1 - \frac{dist(p,p')}{\max Dist(P)}\right) \right]}{\sum_{p' \in N(p)} \left(1 - \frac{dist(p,p')}{\max Dist(P)}\right)}$$

The weight is defined by a pairwise closeness function (in km) between the locations.

Feature Extraction and Prediction

Gradient Boosted Trees (GBTs) are chosen as prediction model, for their demonstrated performances in predictive modeling tasks, also in the context of energy forecasting *(Huang et al, Persson et al).*

We compare predictive performances obtained considering different **experimental settings:**

- Noisy data
- Repaired data
- Repaired data + Feature Extraction

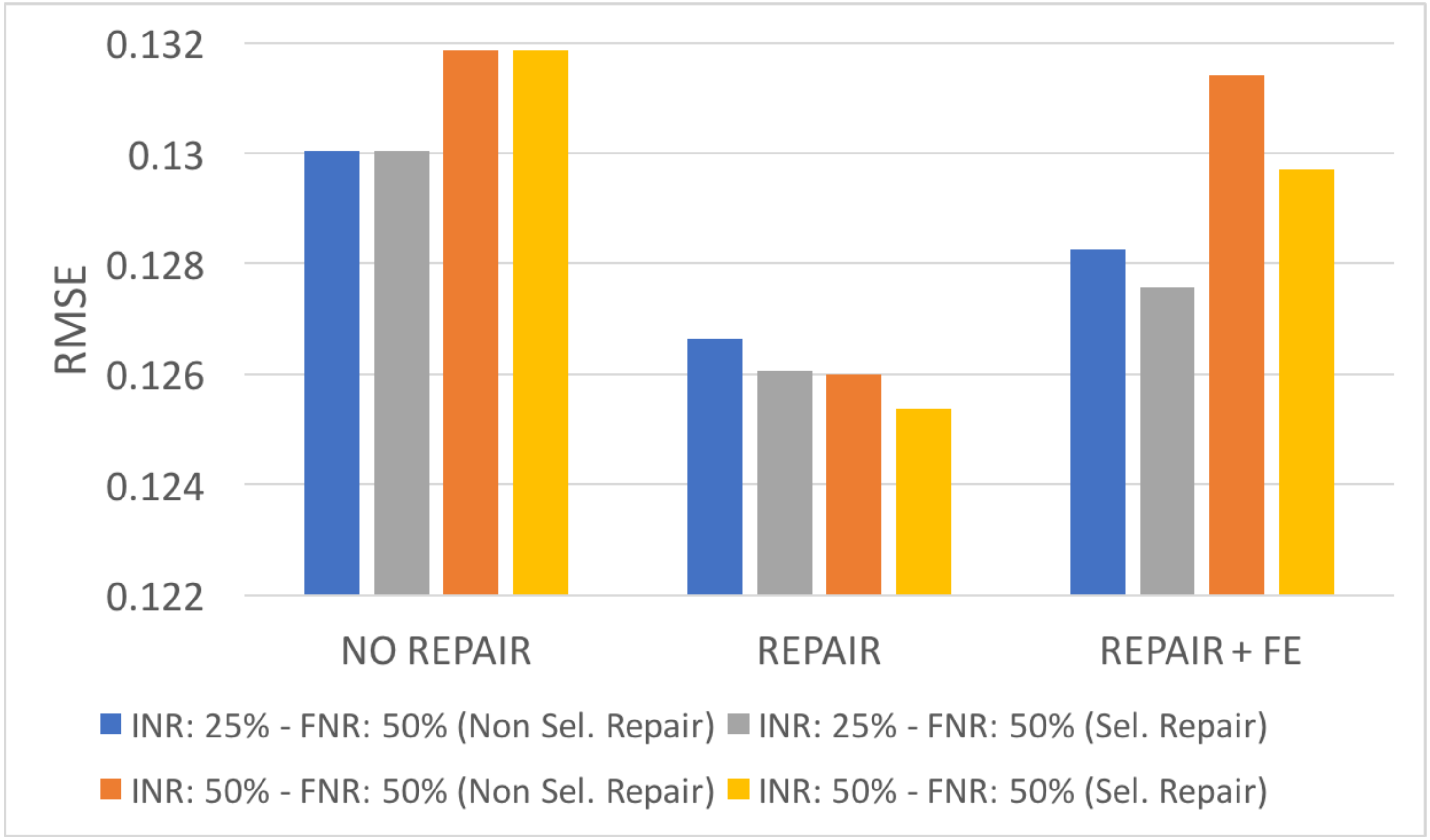
Two Noise Levels:

{25%, 50%} x {Instance Noise Rate (***INR***), Feature Noise Rate (***FNR***)}

Feature extraction is performed exploiting the ***encoding*** function of the Auto-Encoder trained beforehand.

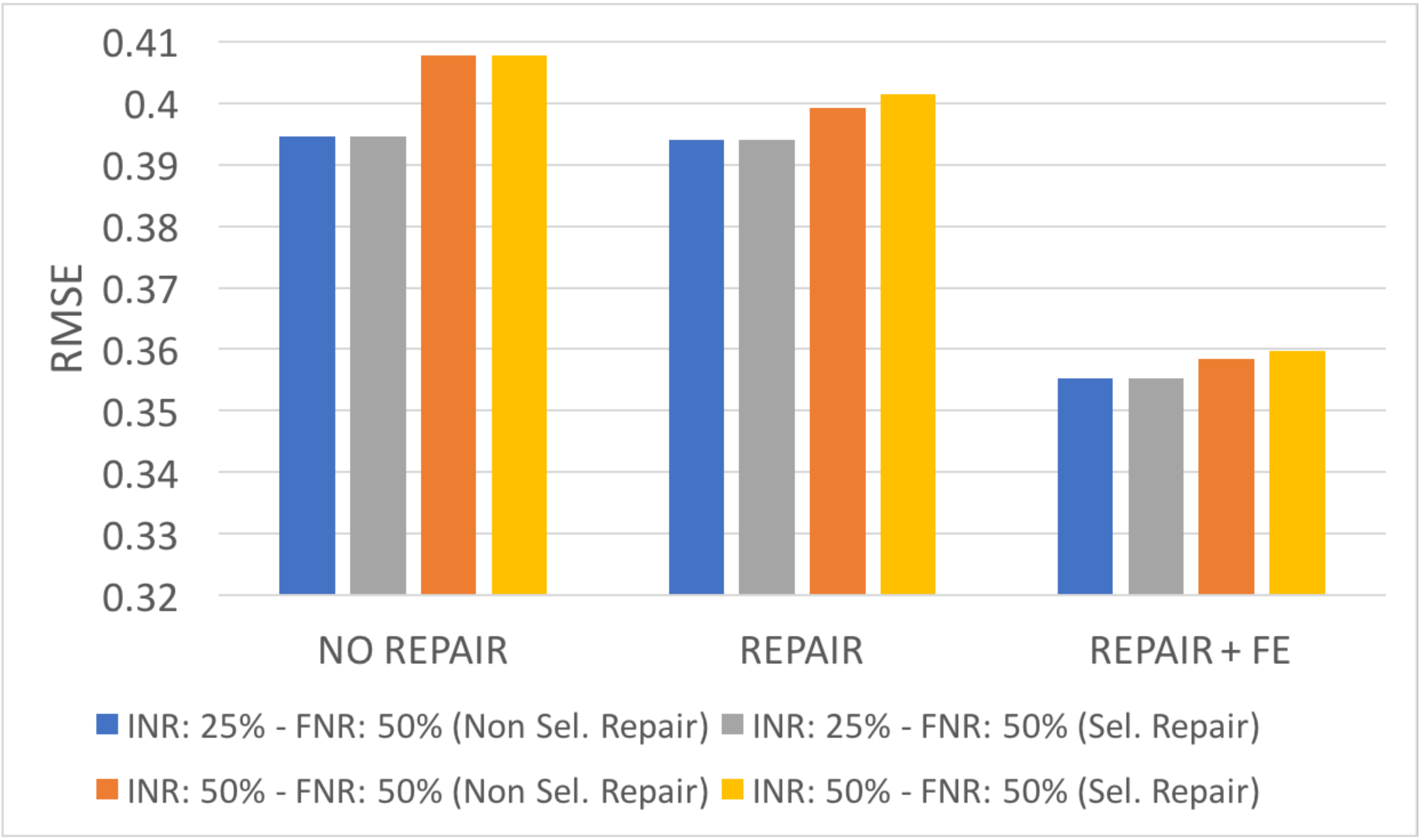
Experimental Results

One-day-ahead power forecasting
PV Italy dataset



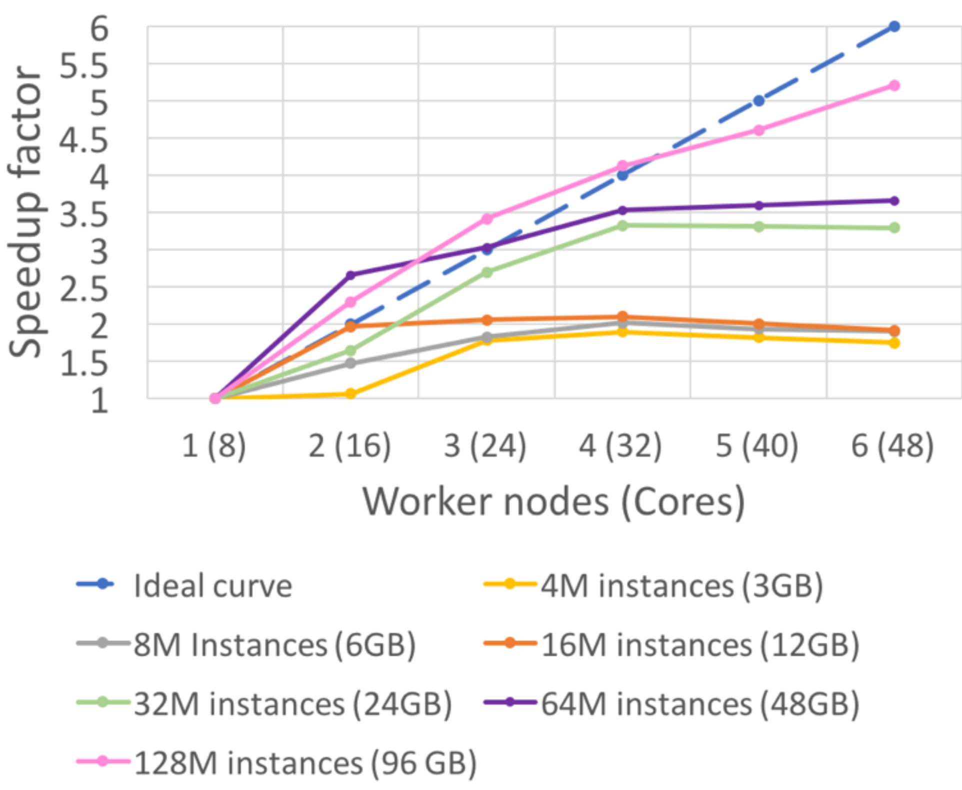
Experimental Results

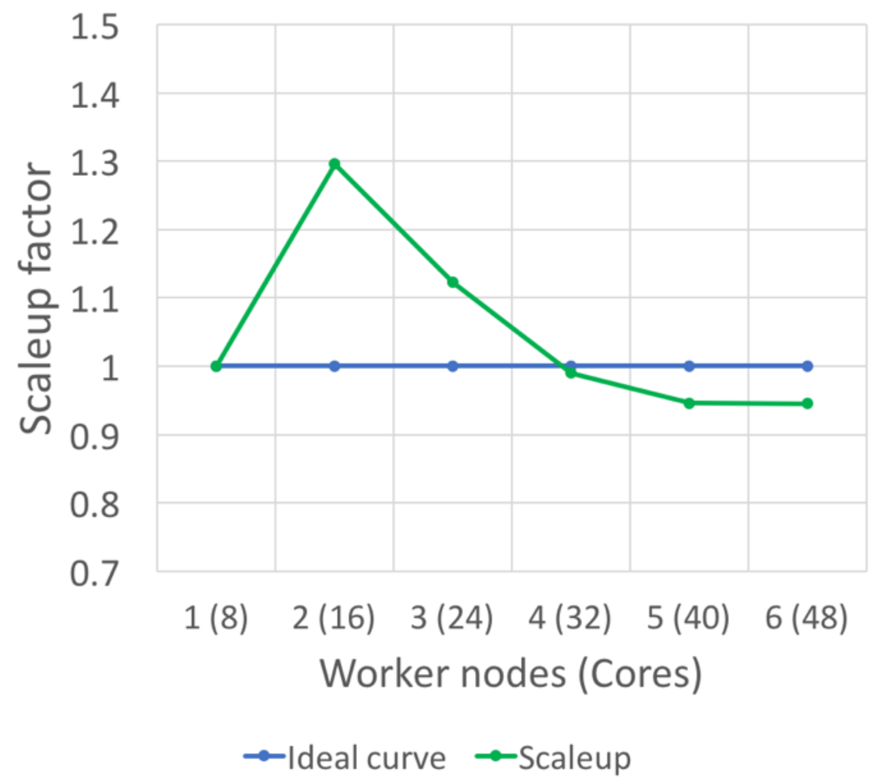
One-day-ahead power forecasting
Wind NREL dataset



Experimental Results

Scalability





Long Short-Term Memory neural networks: overview, possible tasks and applications

LSTM Neural Networks
Overview

- Long Short-Term Memory networks (LSTM) are Recurrent Neural Networks that use memory to process sequences of inputs, and are capable of learning long-term dependencies.

- LSTM prevent backpropagated errors from vanishing or exploding.

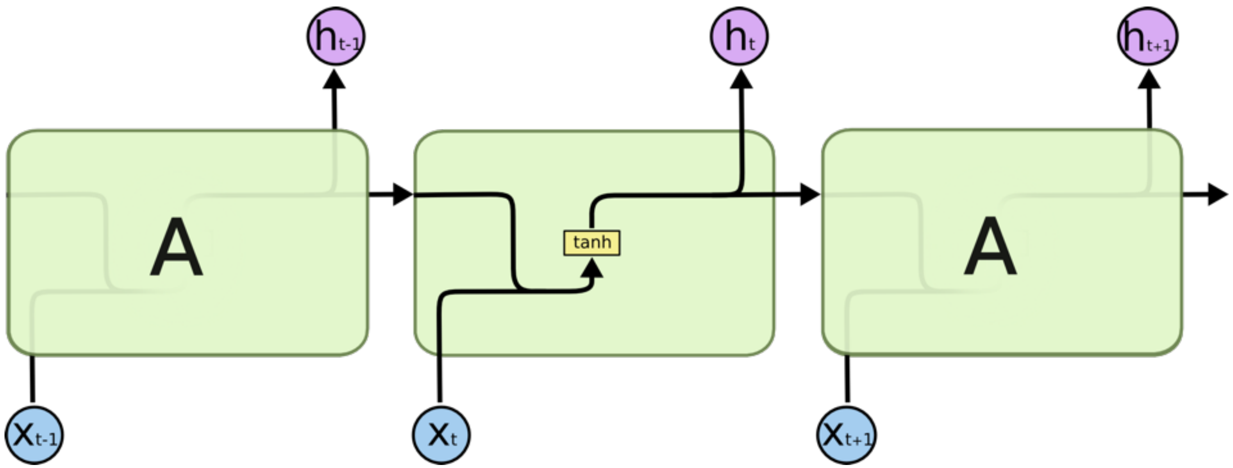
- All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.

- LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.

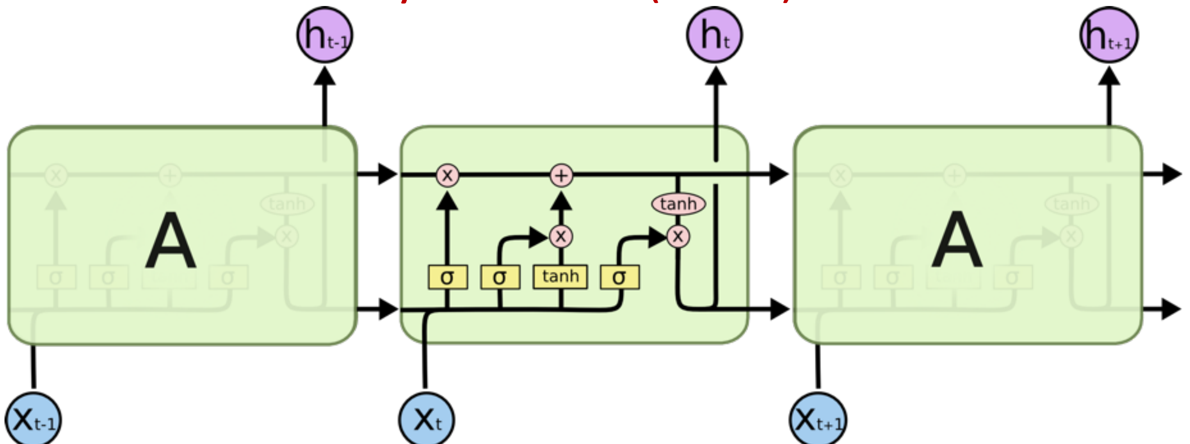
LSTM Neural Networks

Overview

- Recurrent Neural Networks (RNN)



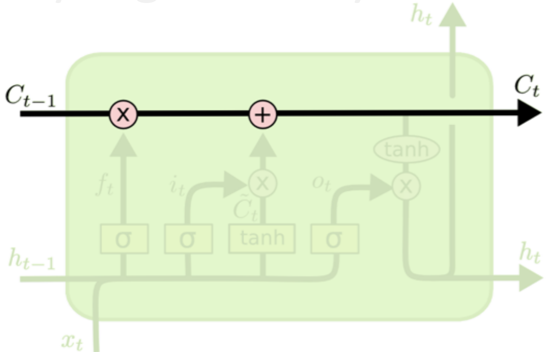
- Long Short-Term Memory networks (LSTM)

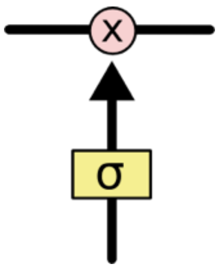


https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM Neural Networks
Overview

- Cell state: It runs straight down the entire chain, with only some minor linear interactions.
- The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.

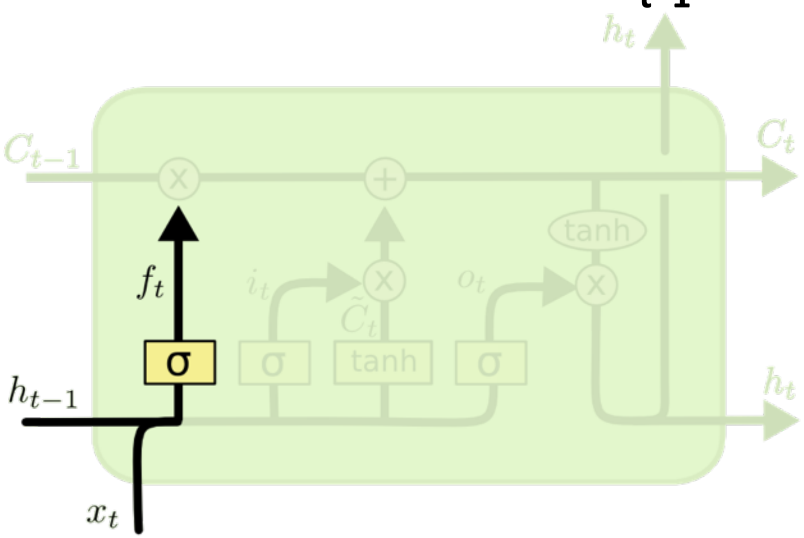




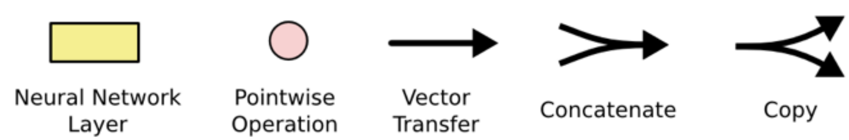
- Gates: A way to optionally let information through. They are composed out of a **sigmoid neural net layer** and a **pointwise multiplication** operation.
- The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through.
- An LSTM has three of these gates, to protect and control the cell state.

LSTM Neural Networks Overview

- The first step in our LSTM is to decide what information has to be discarded from the cell state.
- This decision is made by a sigmoid layer called the "**forget gate layer**"
- It looks at $\mathbf{h_{t-1}}$ and $\mathbf{x_t}$, and outputs a number between 0 and 1 for each number in the cell state $\mathbf{C_{t-1}}$.



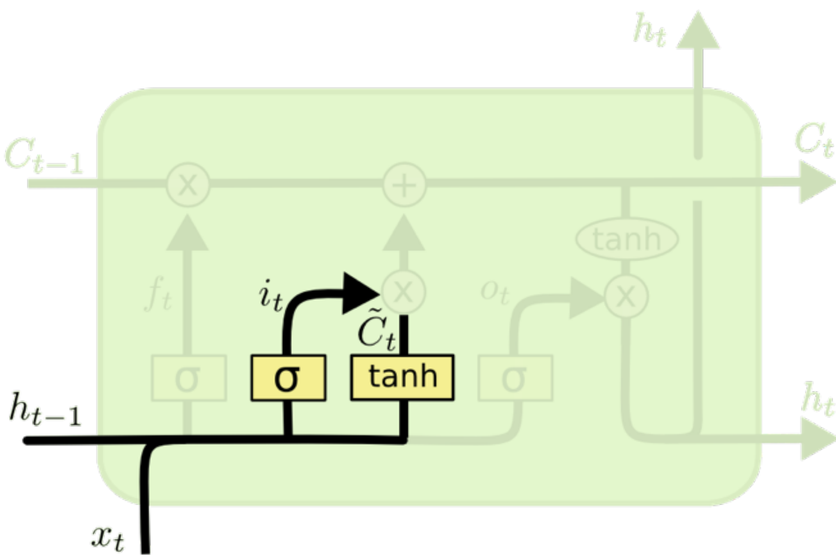
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$



https://colah.github.io/posts/2015-08-Understanding-LSTMs/

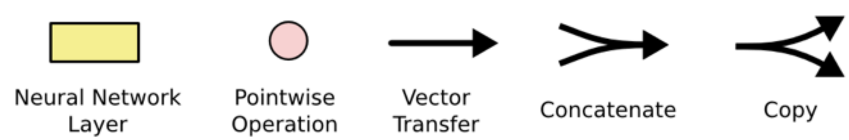
LSTM Neural Networks Overview

- The next step is to decide what new information to store in the cell state.
- First, a sigmoid layer called “**input gate layer**” decides which values to update.
- Next, a **tanh layer** creates a vector of new candidate values, $\tilde{C}_t$, that could be added to the state.
- These two are combined to create an update to the state.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

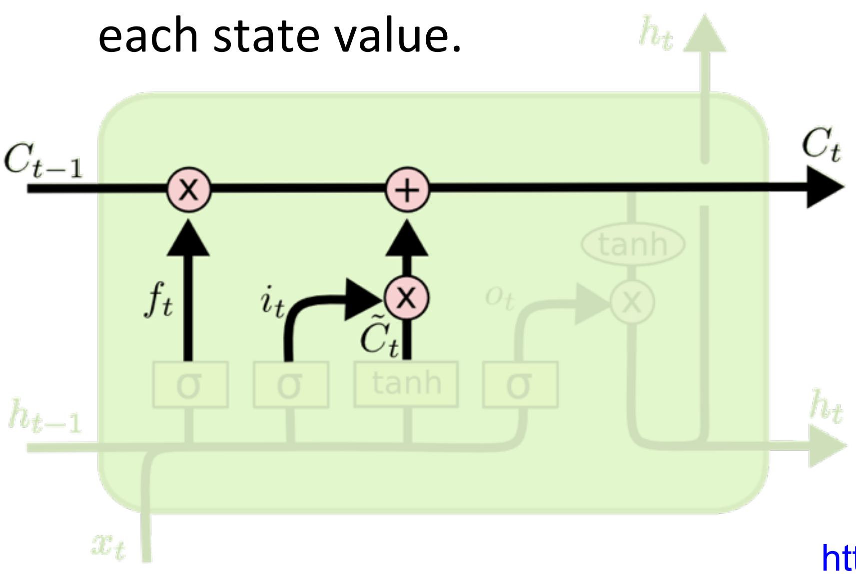
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$



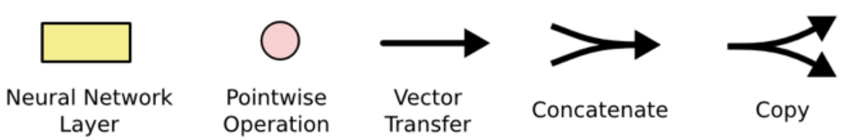
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM Neural Networks Overview

- The old cell state, $\mathbf{C_{t-1}}$ is updated into the new cell state $\mathbf{C_t}$. The previous steps already decided what to do, we just need to actually do it.
- We multiply the old state by $\mathbf{f_t}$, forgetting the things we decided to forget earlier. Then we add $\mathbf{i_t * \tilde{C}_t}$.
- This is the new candidate values, scaled by how much we decided to update each state value.



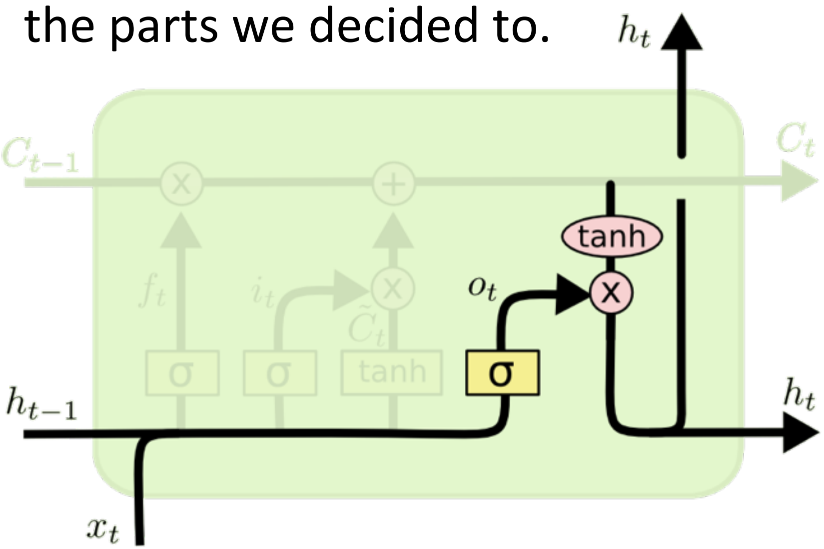
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$



https://colah.github.io/posts/2015-08-Understanding-LSTMs/

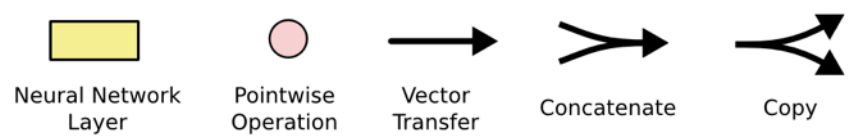
LSTM Neural Networks Overview

- Finally, the output will be based on a filtered version of the cell state.
- First, we run a **sigmoid layer** which decides what parts of the cell state we're going to output.
- Then, we put the cell state through **tanh** to push the values to be between −1 and 1, and **multiply** it by the output of the sigmoid gate, so that we only output the parts we decided to.



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$



https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM Neural Networks in Keras

Code example in Python

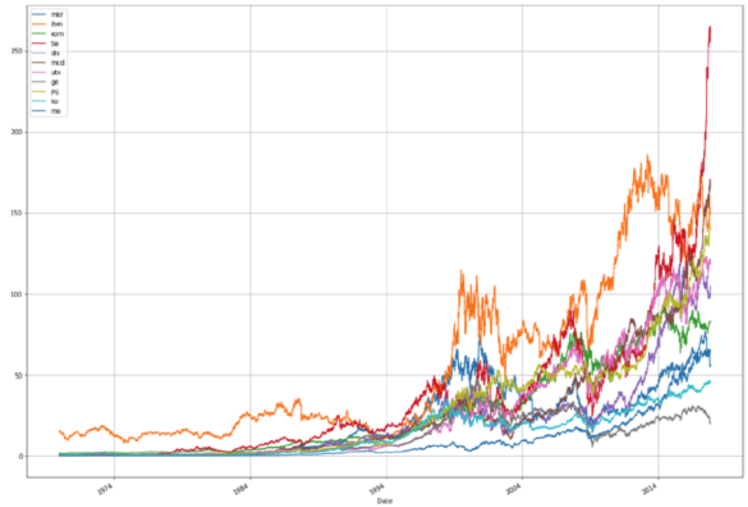
```
def __create_model(self, dim1, dim2):
    self.model = Sequential()
    self.model.add(LSTM(128, input_shape=(dim1, dim2)))
    self.model.add(Dropout(0.2))
    self.model.add(Dense(1))
    self.model.compile(loss='mse', optimizer='adam')

def train(self, x_train, y_train, x_test, y_test, batch_size, test_date):
    self.__create_model(x_train.shape[1], x_train.shape[2])
    history = self.model.fit(x_train, y_train, epochs=15, validation_data=(x_test, y_test), shuffle=False)
    # plot history
    plt.plot(history.history['loss'], label='train')
    plt.plot(history.history['val_loss'], label='test')
    plt.legend()
    plt.savefig('vertical_regression_training_{}.png'.format(test_date))

def predict(self, x_test):
    return self.model.predict(x_test)
```

https://keras.io/

LSTM Neural Networks for Stock Market Forecasting

New York Stock Exchange dataset

- **7195** stocks
- Time range: **1970-01-02** to **2017-11-09**.
- Features each stock:
  - Timestamp – Open – Close – High – Low – Adjusted close – Volume



- Selected time range: dal **2010-01-04** al **2017-11-09.**
- **2337** stocks selected. Each time series (stock) with **1978** observations.
- Technical analysis indicators:
  - Simple and Exponential Moving Average
  - Stochastic Oscillator
  - Relative Strength Index
  - Rate of change
  - Momentum

LSTM Neural Networks for Stock Market Forecasting
Experimental Results

- Mean Absolute Error (**MAE**)

LSTM (vertical dataset)	LSTM (horizontal dataset)	VAR Model
0.000146	0.055504	0.000518

- Root Mean Square Error (**RMSE**)

LSTM (vertical dataset)	LSTM (horizontal dataset)	VAR Model
0.009253128	0.233942064	0.022101298

Publications

Energy forecasting

- CECI M, **CORIZZO R**, FUMAROLA F, MALERBA D, RASHKOVSKA A: Predictive modeling of PV energy production: How to Set Up the Learning Task for a Better Prediction? **IEEE Transactions on Industrial Informatics** Vol. 13, Issue 3, June 2017 (DOI: 10.1109/TII.2016.2604758)

- CECI M, **CORIZZO R**, MALERBA D, RASHKOVSKA A: Spatial Autocorrelation and Entropy for Renewable Energy Forecasting. **Data Mining and Knowledge Discovery** (2019) – in press (Special Issue on Data Mining for Geosciences - DOI: 10.1007/s10618-018-0605-7)

Distributed Growing Self-Organizing Maps

- MALONDKAR A, **CORIZZO R**, KIRINGA I, CECI M, JAPKOWICZ N: Spark-GHSOM: Growing Hierarchical Self Organizing Map for Large Scale Mixed Attribute Datasets **Information Sciences** (2018)

Anomaly detection, repair and feature extraction in smart grids

- **CORIZZO R**, CECI M, JAPKOWICZ N: Anomaly Detection and Repair for Accurate Predictions in Geo-Distributed Big Data. **Big Data Research** (under review)

Thanks for your attention