INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

Departamento de Matemáticas y Física

Desarrollo empresarial, economía social y emprendimiento

PROYECTO DE APLICACIÓN PROFESIONAL (PAP)

4J01 PROGRAMA DE MODELACIÓN MATEMÁTICA PARA EL DESARROLLO DE PLANES Y PROYECTOS DE NEGOCIO I

Modelo de Predicción para el Tipo de Cambio (USDMXN)

PRESENTAN

Programas educativos y Estudiantes

Licenciatura en Ingeniería Financiera. Rodrigo Hernández Mota

Profesor PAP: OCEGUERA GOMEZ ABRAHAM

Tlaquepaque, Jalisco, junio de 2017

ÍNDICE

Contenido

REPORTE PAP 2
Presentación Institucional de los Proyectos de Aplicación Profesional 2
Resumen 2
1. Introducción 2
1.1. Objetivos 2
1.2. Justificación 3
1.3 Antecedentes 3
1.4. Contexto 3
2. Desarrollo 3
2.1. Sustento teórico y metodológico 3
2.2. Planeación y seguimiento del proyecto 3
3. Resultados del trabajo profesional 4
4. Reflexiones del alumno o alumnos sobre sus aprendizajes, las implicaciones éticas y los aportes sociales del proyecto 4
5. Conclusiones 6
6. Bibliografía 7
Anexos (en caso de ser necesarios) 7

REPORTE PAP

Presentación Institucional de los Proyectos de Aplicación Profesional

Los Proyectos de Aplicación Profesional (PAP) son una modalidad educativa del ITESO en la que el estudiante aplica sus saberes y competencias socio-profesionales para el desarrollo de un proyecto que plantea soluciones a problemas de entornos reales. Su espíritu está dirigido para que el estudiante ejerza su profesión mediante una perspectiva ética y socialmente responsable.

A través de las actividades realizadas en el PAP, se acreditan el servicio social y la opción terminal. Así, en este reporte se documentan las actividades que tuvieron lugar durante el desarrollo del proyecto, sus incidencias en el entorno, y las reflexiones y aprendizajes profesionales que el estudiante desarrolló en el transcurso de su labor.

Resumen

En este proyecto se desarrolló un modelo de predicción del tipo de cambio basado en modelos matemáticos avanzados. Se procuró automatizar el procesamiento de las variables y la generación del modelo. Para este sentido, se utilizó el lenguaje de programación Python y librerias de machine learning del "estado del arte". Se propuso un algoritmo para encontrar la mejor arquitectura de la red neuronal. Como resultado se obtuvo un modelo cuyo error promedio de estimación para el siguiente día es de $ 0.11 MXN. Se utiliza un método de simulación basado en los errores de los datos de validación para generar una distribución para la estimación del día siguiente. Esto significa que podemos proporcionar intervalos de confianza como base para las estrategias financieras de cobertura.

1. Introducción

1.1. Objetivos

Generación de un modelo matemático de predicción para el tipo de cambio como insumo clave en el desarrollo de estrategias de cobertura.

1.2. Justificación

Las pequeñas y medianas empresas tienen un alto índice de riesgo e incertidumbre generado por variables financieras externas a su operación. En este proyecto de aplicación profesional se usaron modelos matemáticos avanzados para pronosticar estas variables y proponer estrategias de cobertura con el fin de inmunizar la incertidumbre y optimizar recursos. Esto proporciona a las empresas oportunidades de enfoque y mejora en sus procesos internos y actividades comerciales sin tener que preocuparse de variables macroeconómicas que podrían afectar su proceso de negocio.

1.3 Antecedentes

Los riesgos generados por variables financieras suelen pasar desapercibidos hasta el momento en el cual generan una situación adversa para la empresa en consideración. Una estrategia de cobertura de bajo costo que permite inmunizar el riesgo y optimizar recursos requiere un alto nivel de conocimiento de mercado y gran pericia en los temas financieros. Aunado a esto, las pequeñas y medianas empresas suelen ser más sensibles a la realización de estos riesgos ya que su operación puede verse gravemente afectada por tales variables económicas. Debido a esta necesidad, nos enfocamos a desarrollar el modelo de predicción que permite generar estrategias de cobertura para tales empresas.

1.4. Contexto

La situación económica del país es estable. Sin embargo, recientes acontecimientos internacionales han afectado variables financieras clave para algunos sectores del país. El área de especialización de este proyecto se enfocó en analizar y proponer coberturas respecto al tipo de cambio y las tasas de interés. En este sentido, actualmente existen

fuertes factores políticos en el extranjero (principalmente Estados Unidos) que generan gran incertidumbre.

2. Desarrollo

2.1. Sustento teórico y metodológico

Para generar una estrategia de cobertura financiera es necesario tener una noción del comportamiento de la variable económica en consideración durante el intervalo de tiempo en donde podría representar un riesgo. Un modelo matemático predictivo permite estimar el comportamiento de tal variable en el periodo deseado. En este proyecto nos enfocamos a desarrollar el modelo de predicción.

En la literatura clásica existen diferentes modelos econométricos y estadísticos para estimar variables financieras. No obstante, debido al acelerado avance tecnológico se han desarrollado distintas metodologías y herramientas provenientes del área de "Aprendizaje de Máquina" que permiten innovar en los modelos de predicción convencionalmente usados en el área financiera. Siguiendo esta tendencia, se trabajó en un modelo basado en redes neuronales (perceptrón multicapa) cuya ventaja implica el poder "aprender" de los patrones financieros de forma no lineal. La innovación reside en determinar las variables de entrenamiento (propias para cada problema) y la tecnología y algoritmos usados como base.

El desarrollo de este modelo fue hecho en el lenguaje de programación Python y la implementación de los algoritmos de aprendizaje y entrenamiento fueron programados usando TensorFlow; una librería desarrollada por Google que facilita la computación de algoritmos de "Machine Learning" proporcionando un flujo de datos y procesamiento en base a grafos de tensores.

2.2. Planeación y seguimiento del proyecto

Descripción del proyecto

El proyecto consta en desarrollar un modelo que estime el valor del tipo de cambio (USDMXN) para un tiempo futuro. La construcción teórica de este modelo permite pronosticar para un periodo de n-días a futuro, sin embargo, de forma práctica se decidió especializar el desarrollo para atender el caso particular de predicción a 1 día.

El modelo cuenta con dos áreas principales de fortaleza. La primera consiste en la metodología de detección de variables de entrada para el modelo y la segunda en los métodos de optimización para la arquitectura y en la naturaleza del modelo (algoritmos de aprendizaje).

Plan de trabajo

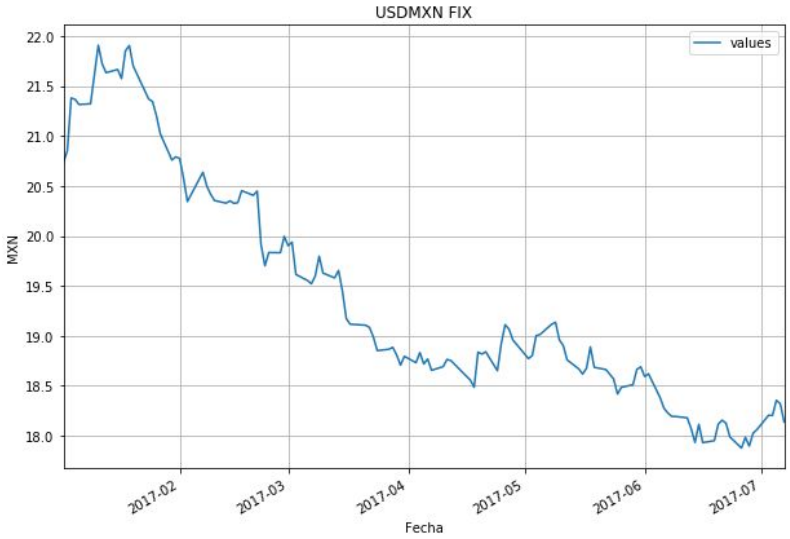
El desarrollo del modelo consiste en las siguientes etapas:

1. Análisis de variable dependiente.
2. Selección y generación de variables independientes
3. Desarrollo del modelo
4. Optimización de arquitectura
5. Generación de estimaciones

Análisis de variable dependiente

La variable a predecir es el tipo de cambio USDMXN, particularmente el valor FIX publicado por el Banco de México cada día laboral a partir de las 12h00. Esta variable financiera hace referencia al promedio de cotizaciones en el mercado cambiario al mayoreo acumuladas hasta su hora de publicación.

Se usaron datos históricos desde 01/01/2017 hasta 10/07/2017. Esta decisión se tomó de tal forma para evitar sesgos de mercado y alta volatilidad presentados a finales del año pasado causados por eventos "únicos" de alta magnitud (presidencia de EEUU).



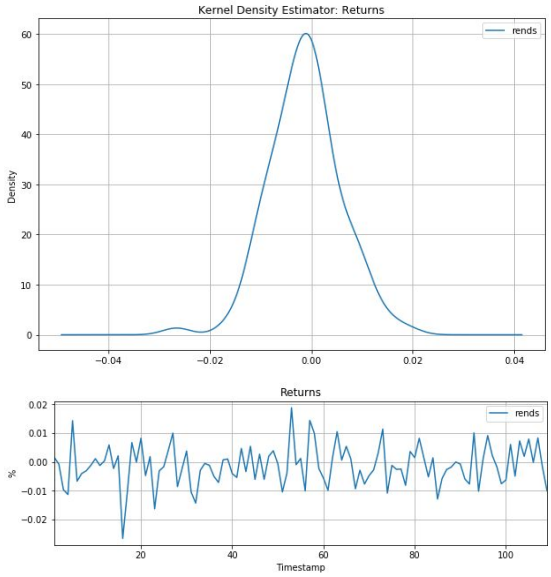
[Figura 01: Valores históricos del tipo de cambio USDMXN **]**

Selección y generación de variables independientes

Las variables usadas en el modelo de predicción fueron seleccionadas en base a teoría económica y correlación con la variable independiente.

Rezagos de precios y rendimientos

La teoría económica del mercado eficiente sugiere que la información relevante de un activo financiero está contenida en su precio. De esta forma, adquiere sentido pensar en modelos auto-regresivos. En econometría tradicional se han desarrollado distintos métodos para determinar el número de rezagos convenientes a usar en modelos de series de tiempo. Para este caso en particular, se usaron los rezagos del tipo de cambio y sus rendimientos usando la función de autocorrelación parcial. Los resultados indican que se debería usar 1 rezago para precios y rendimientos.



[Figura 02: Rendimientos y su estimación de densidad **]**

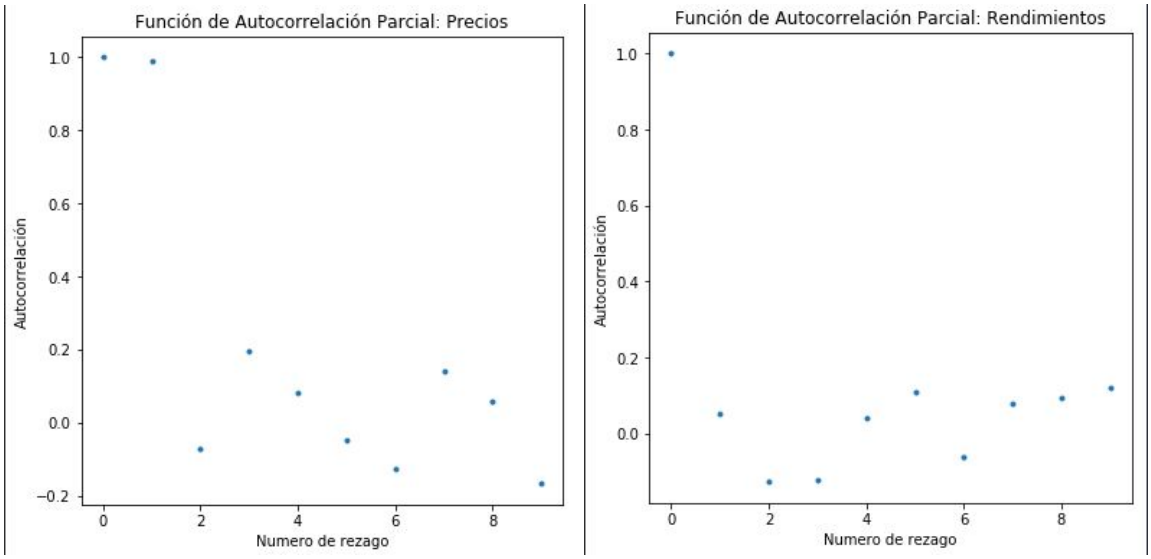
$$x = (x_0, x_1, x_2, \dots, x_{n-k})$$

$$x^k = (x_k, x_{k+1}, x_{k+2}, \dots, x_n)$$

$$FACP_k = a_k$$

$$\rho_k = \frac{E\left[(x-\mu_x)(x^k-\mu_{x^k})\right]}{\sqrt{E[(x-\mu_x)^2]E[(x^k-\mu_{x^k})^2]}} \qquad \begin{bmatrix} \rho_0 & \rho_1 & \cdots & \rho_{k-1} \\ \rho_1 & \rho_0 & \cdots & \rho_{k-2} \\ \vdots & \vdots & \ddots & \vdots \\ \rho_{k-1} & \rho_{k-2} & \cdots & \rho_{k-k} \end{bmatrix}^{-1} \begin{bmatrix} \rho_1 \\ \vdots \\ \rho_{k-1} \\ \rho_k \end{bmatrix} = \begin{bmatrix} a_1 \\ \vdots \\ a_{k-1} \\ a_k \end{bmatrix}$$

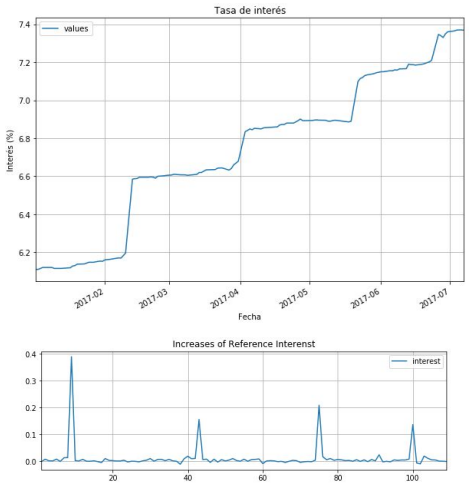
[Figura 03: Definición de autocorrelación y función de autocorrelación parcial **]**



[Figura 04: Determinación de número de rezagos **]**

Diferencias de Tasas de Interés

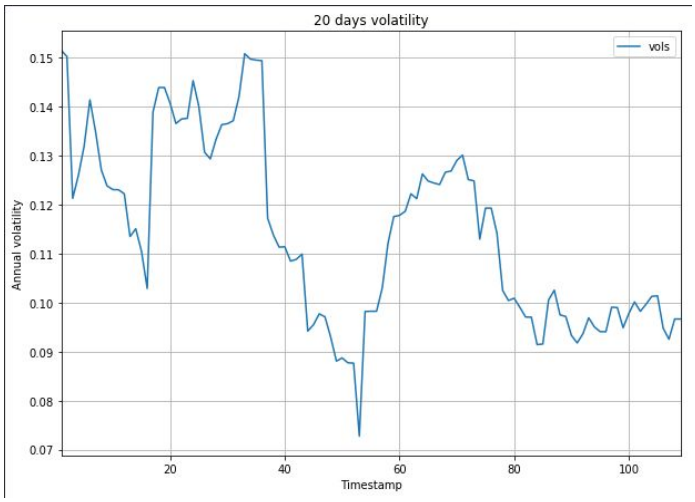
El nivel de tasas de interés de un país tiene implicaciones directas con el tipo de cambio. Esto es debido a que una tasa de interés alta atrae inversión extranjera, lo cual por consecuencia de oferta y demanda aprecia la moneda (el tipo de cambio baja). Bajo esta lógica se propone como variable de entrada las diferencias del nivel de tasa de interés. Para este motivo se utilizó la Tasa de Interés Interbancaria de Equilibrio a plazo de 28 días con publicación diaria por el Banco de México.



[Figura 05: Tasa de interés y diferencias **]**

Volatilidad

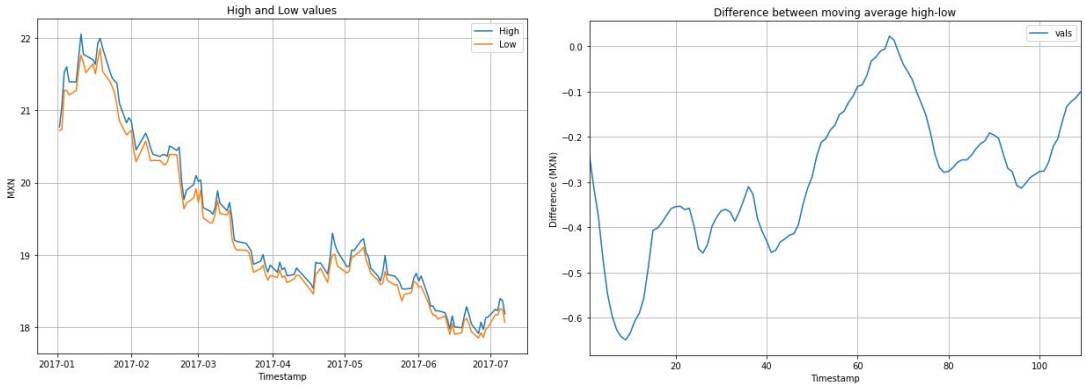
La volatilidad del activo financiero define su comportamiento. Por este motivo, se utiliza la volatilidad anual del tipo de cambio respecto a una ventana de tiempo de 20 días. Esto permite al modelo identificar incrementos o disminuciones en el riesgo / variación del activo. La ventana de tiempo para calcular la volatilidad fue determinada mediante iteraciones en las cuales se escogió aquella que tuviera una mayor correlación con el activo financiero.



[Figura 06: Volatilidad del activo financiero **]**

Diferencia de promedio móviles

El uso de promedios móviles es comúnmente usado como indicador para trading. En este caso se decidió calcular la diferencia de un promedio móvil de corto plazo (10 días) para el min tipo de cambio del día con el promedio móvil de largo plazo (20 días). Si esta diferencia es negativa, significa que se está detectando una tendencia a la baja (min corto plazo < max largo plazo). Por otra parte, si la diferencia es positiva entonces se estaría detectando una subida pronunciada (min corto plazo > max larga plazo).



[Figura 07: Máximo y mínimo del tipo de cambio y diferencias en promedio móvil **]**

Desarrollo del modelo

El modelo de predicción fue desarrollado en el lenguaje de programación Python. Esto es debido a que tal lenguaje nos permite expresar formulación matemática de forma eficiente y crear sistemas complejos. En particular se desarrolló la lógica de descarga y obtención de datos en un servidor público en la plataforma de Heroku. Esto nos permite externalizar el procesamiento computacional de descarga y transformación de datos. Para el entrenamiento se utilizó un servidor privado, lo cual nos permite concentrar los recursos computacionales directamente en el modelo.

Aparte de la generación de variables, existen dos componentes principales en la ingeniería del modelo. Estos componentes son la clase para manejo de datos y la clase que contiene la lógica del perceptrón multicapa. El código para ambos puede ser encontrado en los anexos o en el siguiente repositorio de github: https://github.com/mxquants/quanta. El código fue desarrollado durante este proyecto y está disponible como "open-source".

La clase para el manejo de datos es denominada **Dataset** dentro de **dataHandler.py**. Dentro de la lógica del código se encuentra todo lo necesario para procesar los datos. En el modelo se utilizó en método de normalización "minmax" el cual normaliza las variables de entrada a un valor en el intervalo entre 0 y 1. La otra funcionalidad relevante reside en

la automatización de selección de datos de entrenamiento y prueba en la cual se eligió 70% para entrenar. En los datos de prueba se implementa un algoritmo el cual selecciona x-proporción (lotes) de tales datos de forma aleatoria para cada época de entrenamiento requerida por el modelo. Esto implica que para cada época de entrenamiento el modelo no obtendrá exactamente la misma información.

La clase que contiene la lógica del modelo es denominada **mlpRegressor** dentro de **neuralNets.py**. Este objeto utiliza elementos de la librería de google para machine learning llamada TensorFlow. Esto genera un modelo eficiente y altamente configurable. En la construcción de la red neuronal se considera una arquitectura flexible en la cual se puede determinar la función de activación para cada neuronal dentro de cualquier capa. En la implementación particular de este modelo se decidió utilizar la función sigmoide dentro de las neuronas en las capas ocultas y la función f(x) = x en la capa de salida. Por consecuencia, en las capas ocultas se genera una transformación no lineal y en la capa de salida se ajusta el resultado para estimar una regresión.

Optimización de arquitectura

Teniendo los elementos básicos y las abstracciones necesarias para la generación del modelo el siguiente reto consistió en determinar la mejor arquitectura para el perceptrón multicapa. Para atender esta preocupación se constituyó un algoritmo que nos permitía iterar bajo todas las posibles arquitecturas de la red neuronal y escoger aquella que generó el menor error. Este tipo de algoritmos son denominados como "búsqueda exhaustiva" y tienen la desventaja de consumir bastantes recursos computacionales.

Podemos expresar las distintas combinaciones para una red neuronal como una función **f(y,x)** en donde **x** es el número máximo de neuronas por capa {1, 2, ..., x} y **y** es el número de capas ocultas. El número de combinaciones para esta configuración es: **x^y**. Para

atender todas las posibles arquitecturas debemos iterar para cualquier número de capas ocultas entre 0 y **y**. En este sentido el número de combinaciones total para la arquitectura de una red neuronal con **x** neuronas máximo por capas y máximo **y** capas ocultas es: **x^y + x^(y-1) + x^(y-2) + ... + x^0**.

La región de búsqueda del algoritmo se basó en las combinaciones posibles usando máximo 3 capas ocultas con 5 neuronas. Esto genera 156 modelos distintos de los cuales la mejor arquitectura fue **1 capa oculta con 2 neuronas**.

El siguiente algoritmo recursivo implementado en Python 3.x genera todas las combinaciones posibles de capas ocultas para la arquitectura de la red neuronal. Usar la función **getAllPossibleArchitectures(y, x)**.

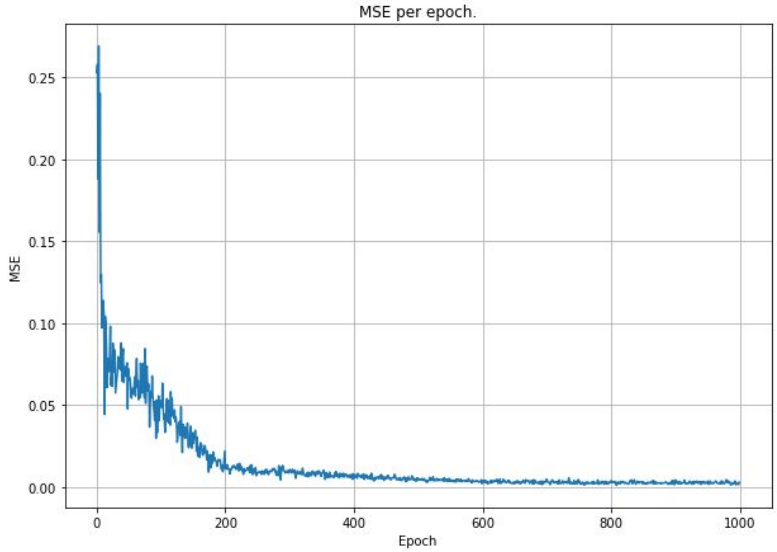
```
def getHiddenCombinations(hidden, max_neurons, count=0):
    """Combination for a particular configuration."""
    if count == max_neurons ** hidden:
        return []
    return [[(lambda i:
             int((count % max_neurons**(i+1)) / max_neurons**i) + 1)(i)
            for i in range(hidden)]] + getHiddenCombinations(hidden,
                                                            max_neurons,
                                                            count=count+1)


def getAllPossibleArchitectures(max_hidden, max_neurons):
    """All hidden layer combinations."""
    if max_hidden == 0:
        return getHiddenCombinations(max_hidden, max_neurons)
    return getHiddenCombinations(max_hidden, max_neurons) + \
        getAllPossibleArchitectures(max_hidden-1, max_neurons)
```

[Figura 08: Algoritmo para obtener todas las combinaciones de arquitectura. **]**

Generación de estimaciones y resultados

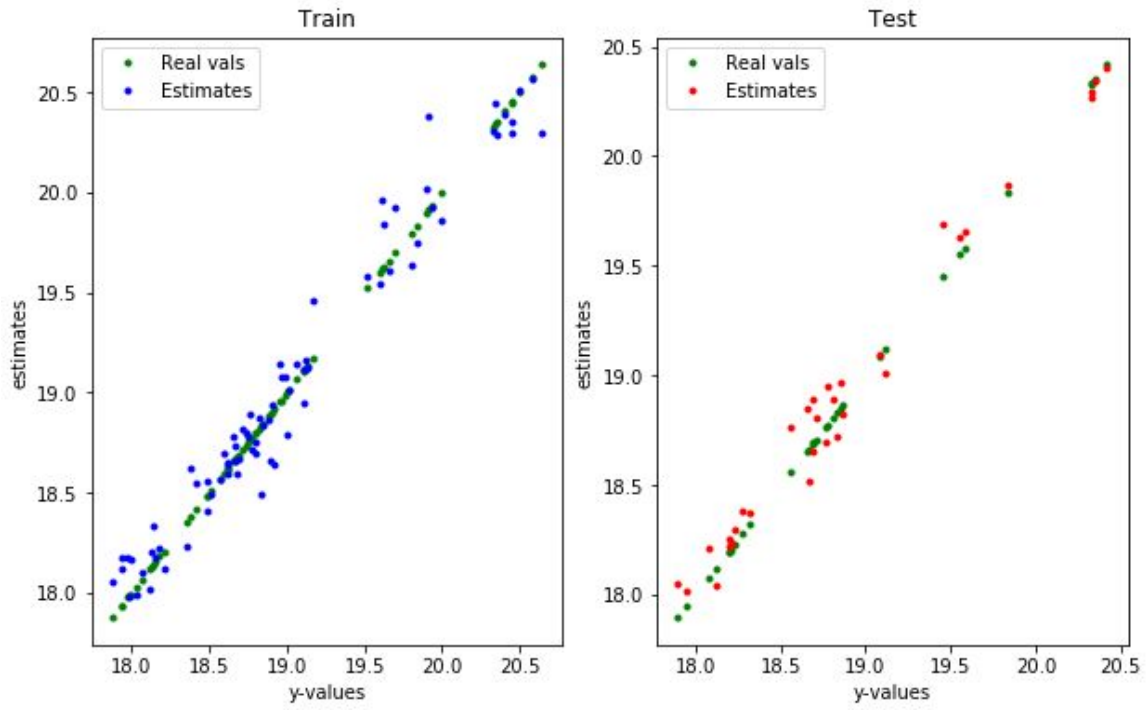
Habiendo determinado la mejor arquitectura de forma exhaustiva, se volvió a entrenar el modelo para generar gráficos ilustrativos de desempeño.

Se utilizaron 1000 épocas de entrenamiento por lotes. La siguiente gráfica muestra el error cuadrado medio del modelo por época.



[Figura 09: Error cuadrado medio por época **]**

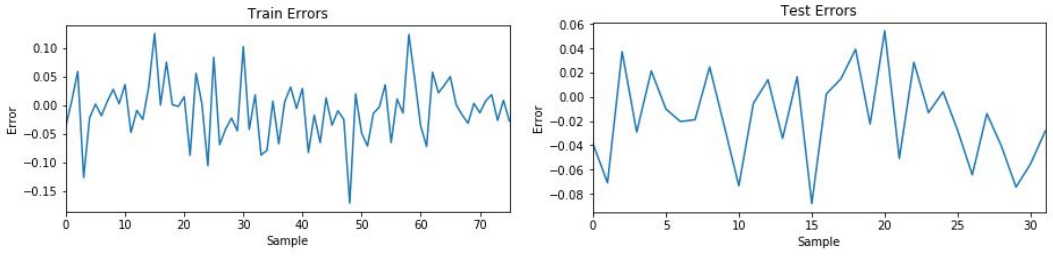
Como se puede observar, el modelo converge rápidamente a una solución. El siguiente gráfico muestra la estimación (desnormalizada) del tipo de cambio con los datos de entrenamiento y prueba.

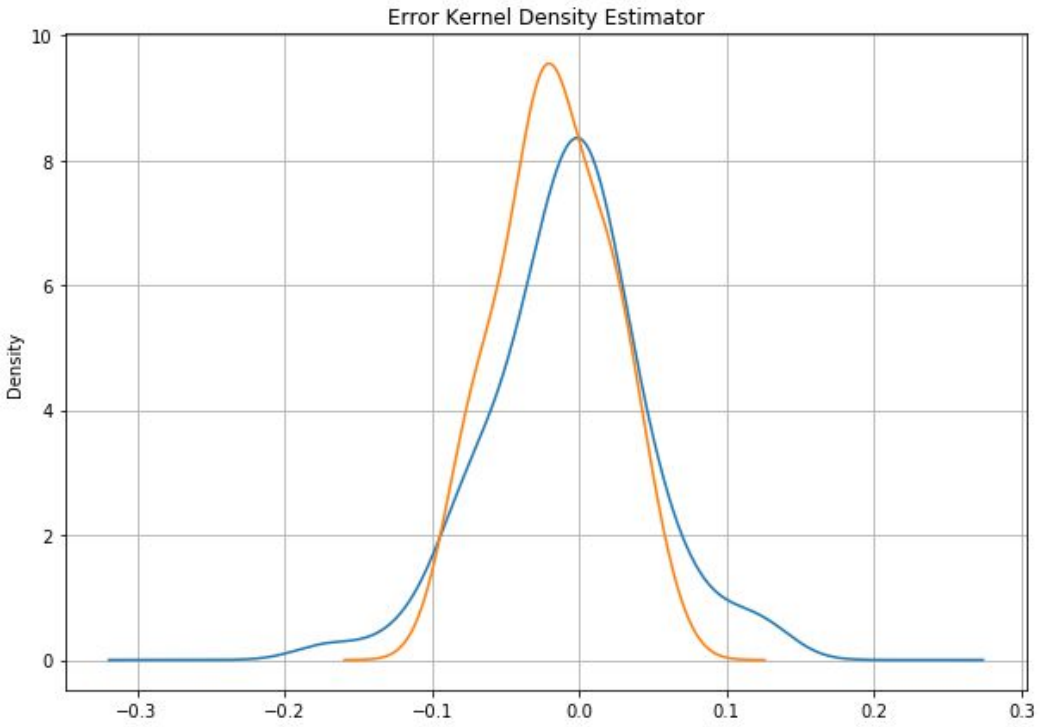


[Figura 10: Estimaciones de datos de entrenamiento y prueba/validación **]**

La interpretación de la figura 10 es relativamente sencilla. Un modelo con estimación perfecta debería sobreponer sus estimaciones tanto de entrenamiento como prueba a los valores reales (indicados de color verde – diagonal). Se puede observar que el modelo tiene cierto nivel de precisión aceptable y tiende a generar buenas estimaciones. La raíz del error cuadrado medio desnormalizado es de **$ 0.14 MXN** y **$ 0.11 MXN** respectivamente lo cual indica que el modelo pudo generalizar bien datos desconocidos (prueba/validación).

Los errores tienen la siguiente forma y distribución:





[Figura 11: Errores y distribución kernel **]**

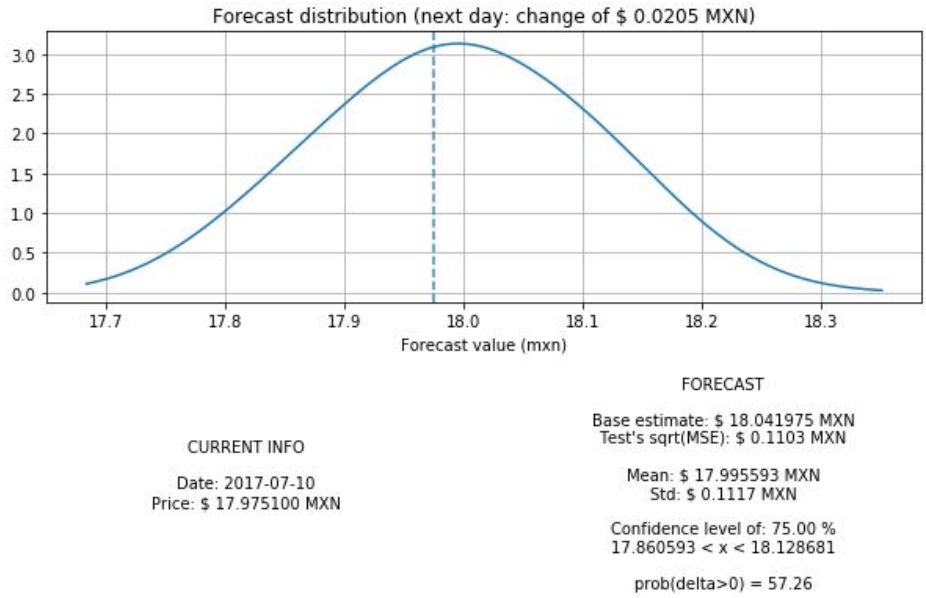
Los errores en ambos casos tienen media cercana a cero y pasaron la prueba de normalidad de Shapiro lo cual indica que el modelo pudo capturar la información relevante del activo financiero. No obstante, se utilizó la distribución empírica generada por kernel gaussianos para generar la distribución de probabilidad de la estimación del tipo de cambio mediante métodos de simulación.

	EstimatedValue	RealValue	RecordedError
0	18.383032	18.2762	-0.106832
1	18.848064	18.6521	-0.195964
2	19.013647	19.1164	0.102753
3	18.889381	18.8092	-0.080181
4	20.266382	20.3254	0.059018
5	18.221719	18.1939	-0.027819
6	18.250996	18.1946	-0.056396
7	18.374889	18.3227	-0.052189
8	18.696859	18.7646	0.067741
9	18.294062	18.2278	-0.066262
10	18.761028	18.5582	-0.202828

[Figura 12: Comparación del tipo de cambio real contra alguna estimaciones (validación) **]**

Ejemplo de estimación

Evaluando el modelo con los datos pertinentes y agregando la distribución de los errores de validación obtenemos la estimación del día siguiente, caso particular al 11/07/2017:



[Figura 13: Pronóstico del tipo de cambio y distribución **]**

En el resultado final mostramos la estimación base (valor en bruto por el modelo) y la estimación promedio (ajustado con el Kernel). Estos valores pueden ser considerados como el pronóstico puntual. No obstante, otra aportación relevante del modelo es proveer la distribución completa de estimación la cual nos permite calcular intervalos de confianza. En el gráfico generamos el intervalo centrado para un 75% de confianza lo cual indica con cuánta certidumbre es posible asegurar que el tipo de cambio se mantenga en tal rango. Aunado a esto, se proporciona la probabilidad de aumento para el siguiente día.

El tipo de cambio real (fix) para el 11/07/2017 fue de **$ 17.9482 MXN**. El error respecto a la estimación promedio fue de **0.0473 MXN** lo cual es consistente con los errores vistos en los datos de validación.

Principales Recursos

Para las etapas de desarrollo anteriormente descritas el principal recurso fue procesamiento computacional. En particular, se necesitó un servidor para la descarga y

almacenamiento de datos de forma automática y otro para el procesamiento y entrenamiento del modelo.

Propuestas de mejora

Aunque el modelo obtuvo resultados aceptables, existen distintas áreas de mejora. Como se ha comentado, el número de datos disponibles para entrenamiento fueron limitados. Esto resultó ser así para evitar algunos acontecimientos que generaron gran volatilidad en el mercado y reacciones inesperadas. Sin embargo, es posible capturar esta información en alguna otra variable de entrada al modelo. Por consecuencia, el área de mejora principal reside en investigar y proponer nuevas variables innovadoras que sirvan como independientes en el modelo.

Se utilizó un algoritmo de búsqueda exhaustiva para encontrar la mejor arquitectura de la red neuronal. Como se mencionó previamente, esto requiere un alto nivel de procesamiento computacional. Se propone mejorar la implementación de búsqueda usando algoritmos heurísticos tales como algoritmos genéticos o "particle swarm optimization". Esto mejoraría el desempeño y reduciría el costo computacional. También nos permitiría explorar más opciones de arquitectura para el modelo.

Siguiendo la lógica del punto de mejora anterior, se deberían considerar distintos tipos de redes neuronales (LSTM: long short term memory, narx, etc) con el fin de encontrar el mejor modelo para estimación.

3. Resultados del trabajo profesional

Se obtuvo la metodología automatizada (código) para crear un modelo financiero de predicción para el tipo de cambio con una precisión aceptable. Habiendo ejecutado los algoritmos de búsqueda y entrenamiento se encontró la mejor arquitectura y los pesos involucrados para describir esta variable financiera.

Los resultados particulares del modelo nos permiten sustentar las estrategias financieras en base a una distribución de probabilidad confiable para las realizaciones futuras del activo subyacente. De esta forma podemos inmunizar el riesgo de forma más precisa y sin incurrir en costos elevados de cobertura (e.g. cubrir sólo lo necesario). Esto permite que las empresas puedan enfocarse en sus procesos de producción y giro de negocio sin necesidad de estar expuestas a riesgos financieros externos.

4. Reflexiones del alumno o alumnos sobre sus aprendizajes, las implicaciones éticas y los aportes sociales del proyecto

Aprendizajes profesionales

El trabajo realizado en este proyecto contiene distintas áreas multidisciplinarias de mi carrera; programación, matemáticas, estadística y finanzas. Converger en un modelo que logró implementar cada área a un nivel muy particular fue una experiencia enriquecedora. Este reto puso a prueba y fortaleció mis habilidades en tales áreas.

De forma puntual, utilizar tecnología en el "estado del arte" y observar la oportunidad de innovación que aún existe en el sector financiero (impulsado por la tendencia fintech) me inspiró profesionalmente a mejorar y preparar más para aprovechar esta oportunidad.

Aprendizajes sociales

El principal aprendizaje social se basa en la identificación de sectores empresariales con alta sensibilidad a la fluctuación de variables financieras externas a su operación y su incapacidad de generar estrategias de inmunización a tales riesgos sea por falta de experiencia es tales áreas o por la complejidad del problema. El impacto de estos riesgos financieros puede repercutir gravemente en la situación financiera de estas empresas vulnerables.

Considero que en este proyecto pude contribuir en el fomento y generación de prácticas financieras sanas que no caen en la especulación y permiten a la empresa desarrollarse sin afectar su nivel productivo.

De igual forma mi aportación principal en la generación del modelo de predicción requirió un alto nivel de innovación y creatividad puesto que el área de "machine learning" y particularmente las nuevas tecnologías (TensorFlow) son pocas veces aplicadas para resolver problemáticas financieras de esta magnitud.

Debido a lo anterior, me percibo con más potencial y capacidades de liderar y dirigir un proyecto innovador procurando orientarlo a la mejora social.

Aprendizajes éticos

Las decisiones que deben ser tomadas en la generación de un modelo financiero pueden tener grandes implicaciones. De forma puntual, modelar sin realizar pruebas de consistencia o análisis de errores puede provocar que el modelo en "entrenamiento" sea atractivo pero en práctica genere estimaciones deficientes. En el área financiera estos errores pueden potencialmente generar pérdidas significativas que ponen en riesgo a la empresa.

Con esta experiencia profesional, mi criterio se ha fortalecido y conozco las implicaciones y alcance de mis decisiones.

Aprendizaje personal

El PAP me permitió poner a prueba mis conocimientos, habilidades y pasar de la teoría a la práctica. Generar un sistema automatizado para la predicción del tipo de cambio y ligarlo como insumo para distintas estrategias financieras fue un reto en el cual pude aprender bastante. Gracias a esto pude conocer mejor la industria financiera y su relación con el aspecto social y económico. Por consecuencia, mi proyecto de vida ha adquirido una dirección más firme y una perspectiva más abierta.

5. Conclusiones

Durante el desarrollo de este proyecto pude identificar y poner a prueba mis habilidades en un entorno profesional. Se logró satisfacer los requerimientos y el objetivo principal el cual constaba en generar un modelo de predicción del tipo de cambio. De forma muy

puntual, el uso de tecnología en el "estado del arte" tales como TensorFlow enriqueció el desarrollo del modelo y los aprendizajes generados. Considero que de igual forma se atendió el aspecto social y se transmitió la magnitud e importancia del sentido ético en la profesión.

Los resultados del modelo permitieron justificar y construir estrategias financieras con derivados estructurados con mayor precisión y sustento teórico.

6. Bibliografía

G. R. (2015, November 9). Large-Scale Machine Learning on Heterogeneous Distributed Systems. Retrieved July 11, 2015, from http://download.tensorflow.org/paper/whitepaper2015.pdf

Murphy, K. (n.d.). Machine Learning: A Probabilistic Perspective (Adaptive Computation and Machine Learning series) (1st ed.). doi: 978-0262018029

Banco de México. (n.d.). Retrieved July 11, 2017, from http://www.banxico.org.mx/

Anexos (en caso de ser necesarios)

dataHandler.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""Dataset Handler Module.

Provide function and classes to handle datasets for neuralNets.py module.
@author: Rodrigo Hernández-Mota
rhdzmota@mxquants.com
"""

# Imports
import numpy as np
import pandas as pd


class Dataset(object):
  """Dataset class for handling dataset normalization and splits."""

  def __init__(self, input_data, output_data, normalize=None, datatypes={}):
    """Dataset class for handling dataset normalization and splits."""
    self.input_data = input_data.values if self._isPandasOrSeries(
                                input_data) else input_data
```

```
        self.output_data = output_data.values if self._isPandasOrSeries(
                                    output_data) else output_data
        self.input_data = self._addOneDim(self.input_data)
        self.output_data = self._addOneDim(self.output_data)
        self.norm_input_data, self.norm_output_data = None, None
        self.normalized = normalize
        if normalize:
            if len(datatypes.keys()) == 0:  # create datatypes dict
                datatypes["input_data"] = [1]*np.shape(input_data)[-1]
                datatypes["output_data"] = [1]*np.shape(output_data)[-1]
            if "mean" in normalize:
                df_input = pd.DataFrame(self.input_data)
                self.norm_input_data, self.mu_x, self.std_x = \
                    self._normalizeWithMean(df_input,
                                            datatypes["input_data"])
                df_output = pd.DataFrame(self.output_data)
                self.norm_output_data, self.mu_y, self.std_y = \
                    self._normalizeWithMean(df_output,
                                            datatypes["output_data"])
            if "minmax" in normalize:
                df_input = pd.DataFrame(self.input_data)
                self.norm_input_data, self.min_x, self.max_x = \
                    self._normalizeWithMinMax(df_input,
                                              datatypes["input_data"])
                df_output = pd.DataFrame(self.output_data)
                self.norm_output_data, self.min_y, self.max_y = \
                    self._normalizeWithMinMax(df_output,
                                              datatypes["output_data"])
        datasets = self._splitDataset(0.7)
        self.test, self.train = datasets["test"], datasets["train"]

    def _normalizeWithMean(self, df, typelist):
        """Normalize data using (x-mu)/sigma."""
        vector_mu, vector_std = [], []
        for col, is_numeric in zip(df.columns, typelist):
            _mean = np.mean(df[col].values) if is_numeric else 0
            _std = np.std(df[col].values) if is_numeric else 1
            vector_mu.append(_mean)
            vector_std.append(_std)
        return (((df-vector_mu)/vector_std).values, np.array(vector_mu),
                np.array(vector_std))

    def _normalizeWithMinMax(self, df, typelist):
        """Normalize with minmax."""
        _min, _max = [], []
```

```
    for col in df.columns:
        _min.append(np.min(df[col].values))
        _max.append(np.max(df[col].values))
    _min, _max = np.array(_min), np.array(_max)
    return (df-_min)/(_max-_min), _min, _max

def _isPandasOrSeries(self, df):
    return (type(df) == pd.core.frame.DataFrame) or \
                        (type(df) == pd.core.series.Series)

def _addOneDim(self, data):
    shape = len(np.shape(data))
    return np.asarray([[i] for i in data]) if shape == 1 \
        else np.asarray(data)

def _splitDataset(self, cut=0.7):
    # Length and random index
    n = len(self.input_data)
    _index = np.arange(n)
    np.random.shuffle(_index)
    # Retreive data
    x_data = self.norm_input_data if self.normalized else self.input_data
    y_data = self.norm_output_data if self.normalized else self.output_data
    # convert to dataframe
    x_data, y_data = pd.DataFrame(x_data), pd.DataFrame(y_data)
    # separate into test and train
    _split = np.int(np.asscalar(np.round(n*cut)))
    train_set = (x_data.iloc[_index[:_split]].values,
                 y_data.iloc[_index[:_split]].values)
    test_set = (x_data.iloc[_index[_split:]].values,
                y_data.iloc[_index[_split:]].values)
    return {'test': test_set, 'train': train_set}

def nextBatch(self, batch_size=None, adjust=True):
    """Generate next data batch."""
    n = len(self.train[0])
    if not batch_size:
        adj_size = int(0.7*n) if adjust else 100
        batch_size = np.int(n*0.2) if n > 500 else adj_size
    if batch_size > n:
        print("Error: No enought data (min. batch_size = 100).")
        return None
    sample = np.random.choice(np.arange(n), batch_size)
    return self.train[0][sample], self.train[1][sample]
```

```
def testData():
    """Test Object."""
    x_data = pd.DataFrame({"x1": np.random.rand(5000),
                           "x2": np.arange(5000)**2})
    y_data = x_data.apply(lambda x: np.power(np.sqrt(x[1]), x[0]) + 5 *
                          np.sin(np.pi * x[0]), 1)
    datasets = Dataset(x_data, y_data, normalize="mean")
    return x_data, y_data, datasets
```

neuralNets.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""Neural Networks.

Simple neural networks using tensorflow
@author: Rodrigo Hernández-Mota
rhdzmota@mxquants.com
"""


import numpy as np
import tensorflow as tf


class mlpBase(object):
    """Base guideline for MLPs."""

    def __init__(self, hidden_layers, activation_function=[]):
        """Initialize mlp Regressor."""
        # arguments to lists
        hidden_layers_list = [hidden_layers] if type(hidden_layers) == str \
            else list(hidden_layers)
        activ_funcs_list = [activation_function] if type(activation_function) \
            == str else list(activation_function)

        self.hidden_layers = hidden_layers_list
        if len(hidden_layers_list) == len(activ_funcs_list):
            self.activation = [self._activation(i)
                               for i in activation_function] + \
                                   self._lastActivation(self._type)
        elif len(hidden_layers_list) > len(activ_funcs_list):
            new_activ_func = []
```

```
            for i in range(len(hidden_layers_list)):
                new_activ_func.append("sigmoid")
            self.activation = new_activ_func + self._lastActivation(self._type)
        else:
            print("Legnth mismatch among hidden_layers and activation_" +
                  "function.")

    def _lastActivation(self, _type):
        return [""] if "reg" in _type else ["sigmoid"]

    def _activation(self, _string):
        functions = {"sigmoid": tf.nn.sigmoid,
                     "relu": tf.nn.relu,
                     "softmax": tf.nn.softmax,
                     "": lambda x: x}
        return functions.get(_string)

    def _initilizeModel(self, n_inputs, n_outputs=1):
        self.weights, self.bias = {}, {}
        _c = 0
        last_size = n_inputs
        # no hidden layers
        if not len(self.hidden_layers):
            self.weights[_c] = tf.Variable(tf.zeros([last_size, n_outputs]))
            self.bias[_c] = tf.Variable(tf.zeros([n_outputs]))
        # at least 1 hidden layer
        else:
            for i in self.hidden_layers+[n_outputs]:
                self.weights[_c] = tf.Variable(tf.zeros([last_size, i]))
                self.bias[_c] = tf.Variable(tf.zeros([i]))
                last_size = i  # change "columns" to "rows" for next layer.
                _c += 1  # increase position
        return self.weights, self.bias

    def prediction(self, x, weights, biases):
        """Prediction function."""
        layers = [x]
        for i, z in zip(sorted(weights.keys()), self.activation):
            phi = self._activation(z)
            layers.append(phi(tf.add(tf.matmul(
                                        layers[-1],
                                        weights[i]),
                                biases[i])))
        self.layers = layers
        return layers[-1]
```

```
class mlpRegressor(mlpBase):
    """MultiLayer Perceptron Regressor using tensorflow."""

    # Variables
    _type = "regressor"

    def cost(self, pred, y):
        """Cost function."""
        return tf.reduce_mean(tf.squared_difference(pred, y))

    def train(self, dataset, alpha=0.01, epochs=500):
        """Train the model."""
        self.n_inputs = np.shape(dataset.input_data)[-1]
        self.n_outputs = np.shape(dataset.output_data)[-1]
        y = tf.placeholder(tf.float32, [None, self.n_outputs])
        x = tf.placeholder(tf.float32, [None, self.n_inputs])

        tf_vars = {}
        self.weights, self.biases = self._initilizeModel(self.n_inputs,
                                            self.n_outputs)
        tf_vars["pred"] = self.prediction(x, self.weights, self.biases)
        tf_vars["cost"] = self.cost(tf_vars["pred"], y)
        tf_vars["opt"] = tf.train.AdamOptimizer(alpha).minimize(
                                        tf_vars["cost"])
        tf_vars["mse"] = tf.squared_difference(tf_vars["pred"], y)
        self.epoch_error = []
        self.init_op = tf.global_variables_initializer()
        result = {"y": [np.asscalar(i) for i in dataset.train[-1]]}
        with tf.Session() as sess:
            sess.run(self.init_op)
            for epoch in range(epochs):
                x_batch, y_batch = dataset.nextBatch()
                sess.run(tf_vars["opt"], feed_dict={x: x_batch,
                                        y: y_batch})
                self.epoch_error.append(np.mean(sess.run(tf_vars["mse"],
                                feed_dict={x: x_batch, y: y_batch})))
            vals = sess.run(tf_vars["pred"], feed_dict={x: dataset.train[0]})
            self.np_w = sess.run(self.weights)
            self.np_b = sess.run(self.biases)
            self.tf_vars = tf_vars
        result["estimates"] = [np.asscalar(i) for i in vals]
        result["errors"] = [j-i for i, j in zip(
                            result["estimates"], result["y"])]
```

```
    result["square_error"] = [i**2 for i in result["errors"]]
    self.train_results = result

  def freeEval(self, free_input):
    """Custom input to evaluate."""
    x = tf.placeholder(tf.float32, [None, self.n_inputs])
    weights = self.np_w
    biases = self.np_b
    pred = self.prediction(x, weights, biases)
    self.init_op = tf.global_variables_initializer()
    with tf.Session() as sess:
      sess.run(self.init_op)
      vals = sess.run(pred, feed_dict={x: free_input})
    return vals

  def evaluate(self, dataset):
    """Evaluate train data from a given dataset."""
    x_test = dataset.test[0]
    return self.freeEval(x_test)

  def test(self, dataset):
    """Test data from a dataset object."""
    vals = self.evaluate(dataset)
    real_y = dataset.test[-1]
    result = {"estimates": [np.asscalar(i) for i in vals],
          "y": [np.asscalar(i) for i in real_y]}
    result["errors"] = [j-i for i, j in zip(
                        result["estimates"], result["y"])]
    result["square_error"] = [i**2 for i in result["errors"]]
    self.test_results = result
    return result
```