Supervised Learning

Neural Networks and Deep Learning

Peerapon S.

Machine Learning

Data and codes : https://kmutt.me/CPE-ML

Topics

Neural networks basics

- Activation function
- Cost and Loss function
- Gradient descent algorithm

Training neural networks

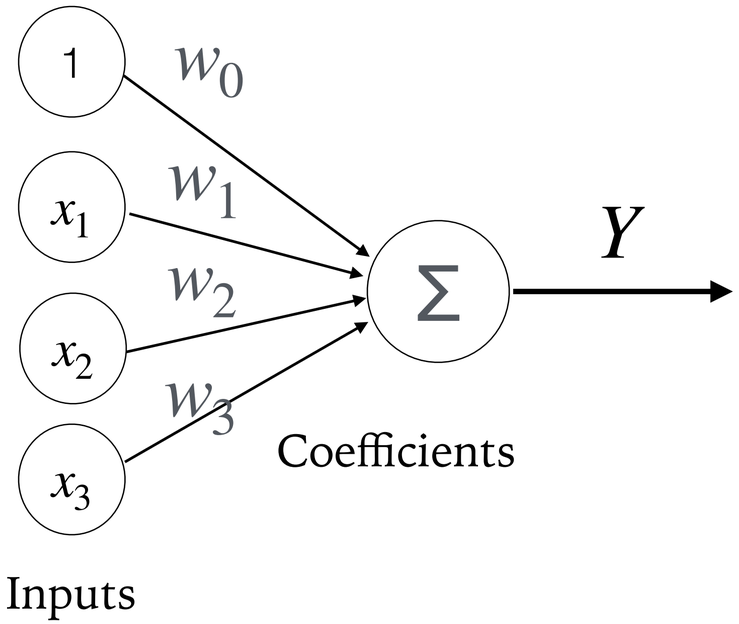
Regression model implementation using Keras

Neural networks for classification problems

Avoiding overfitting

Linear Regression Model

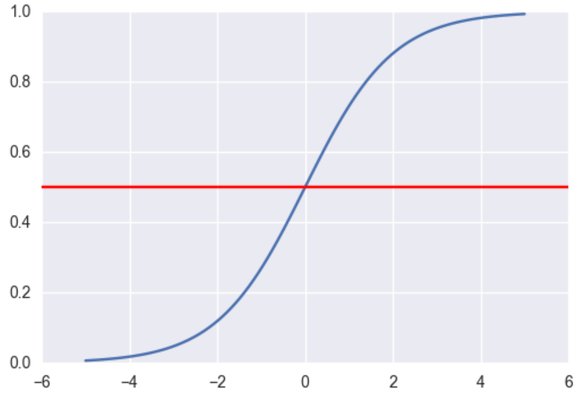
$$Y = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3$$

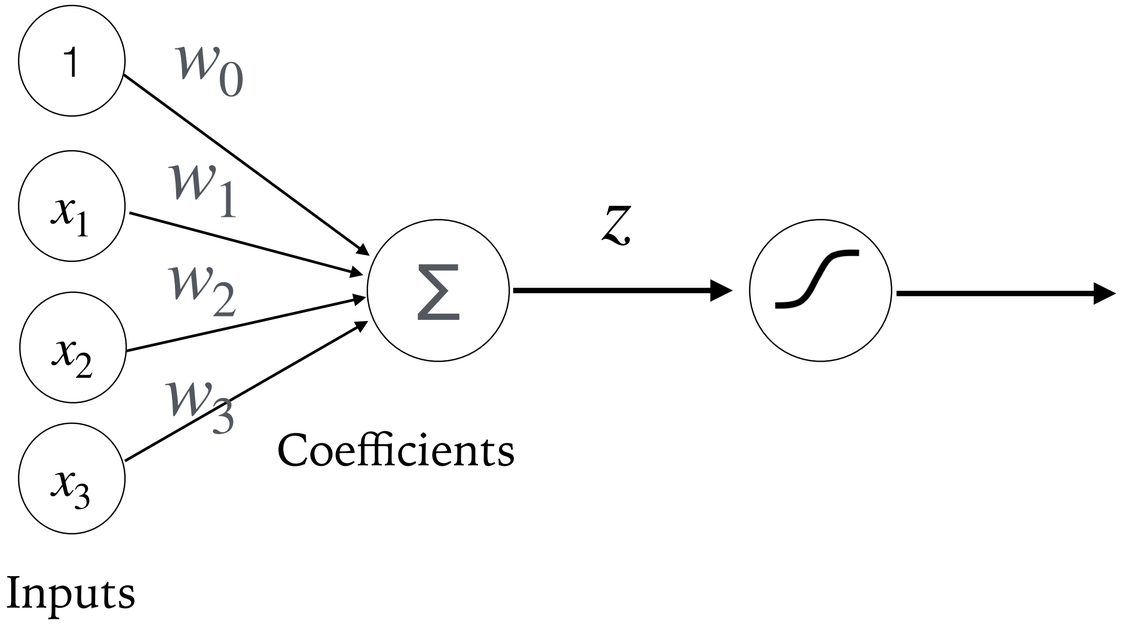


Logistic Regression Model

$$z = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3$$

$$Y = \frac{1}{1 + e^{-z}}$$



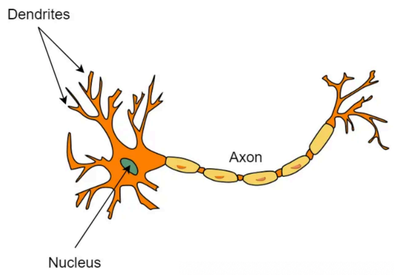


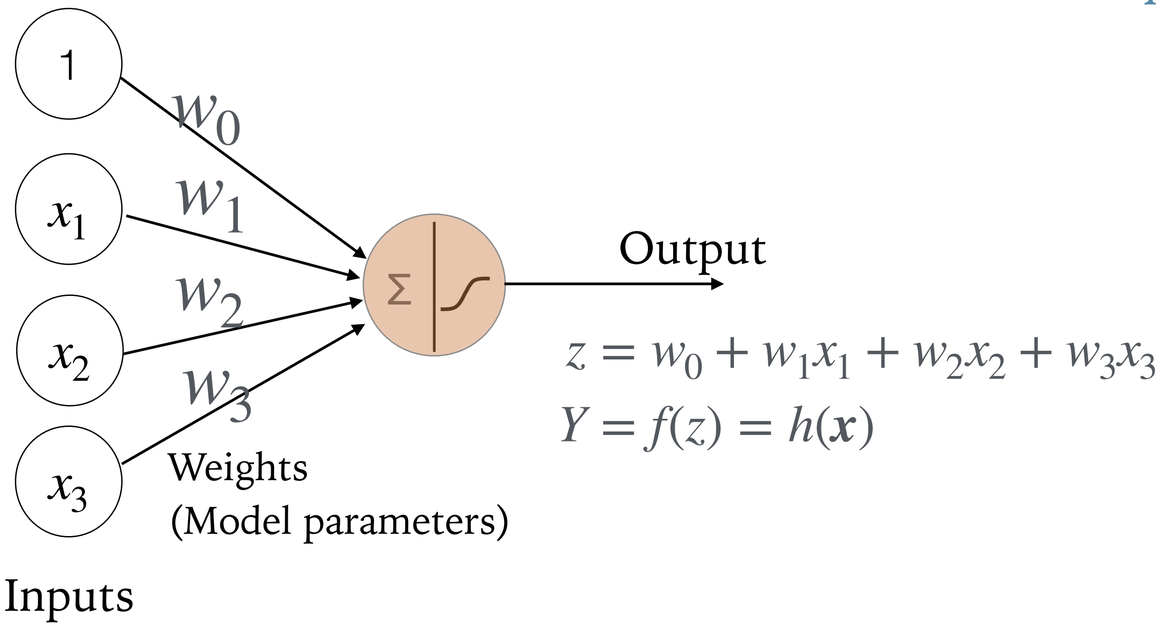
$$Y = f(z) = \frac{1}{1 + e^{-z}}$$

Artificial Neuron

Aka a (single-layer) perceptron, a single-layer (single-output) neural network.

Can have several neurons in the output layer.





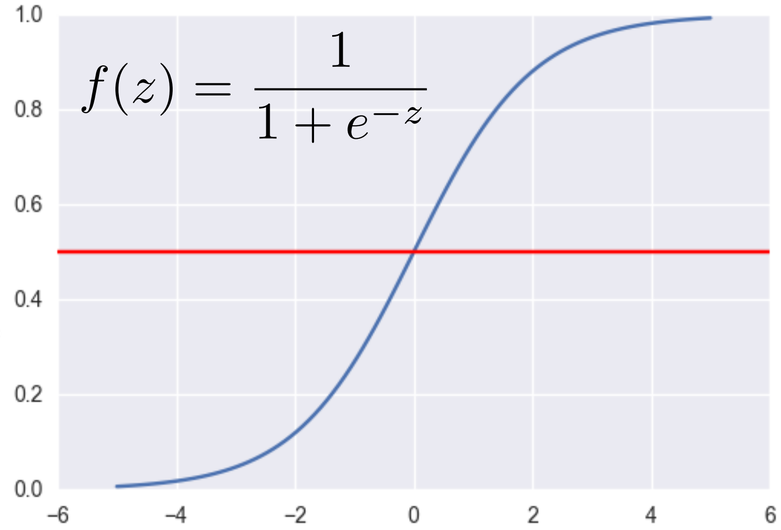
$$z = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3$$

$$Y = f(z) = h(\boldsymbol{x})$$

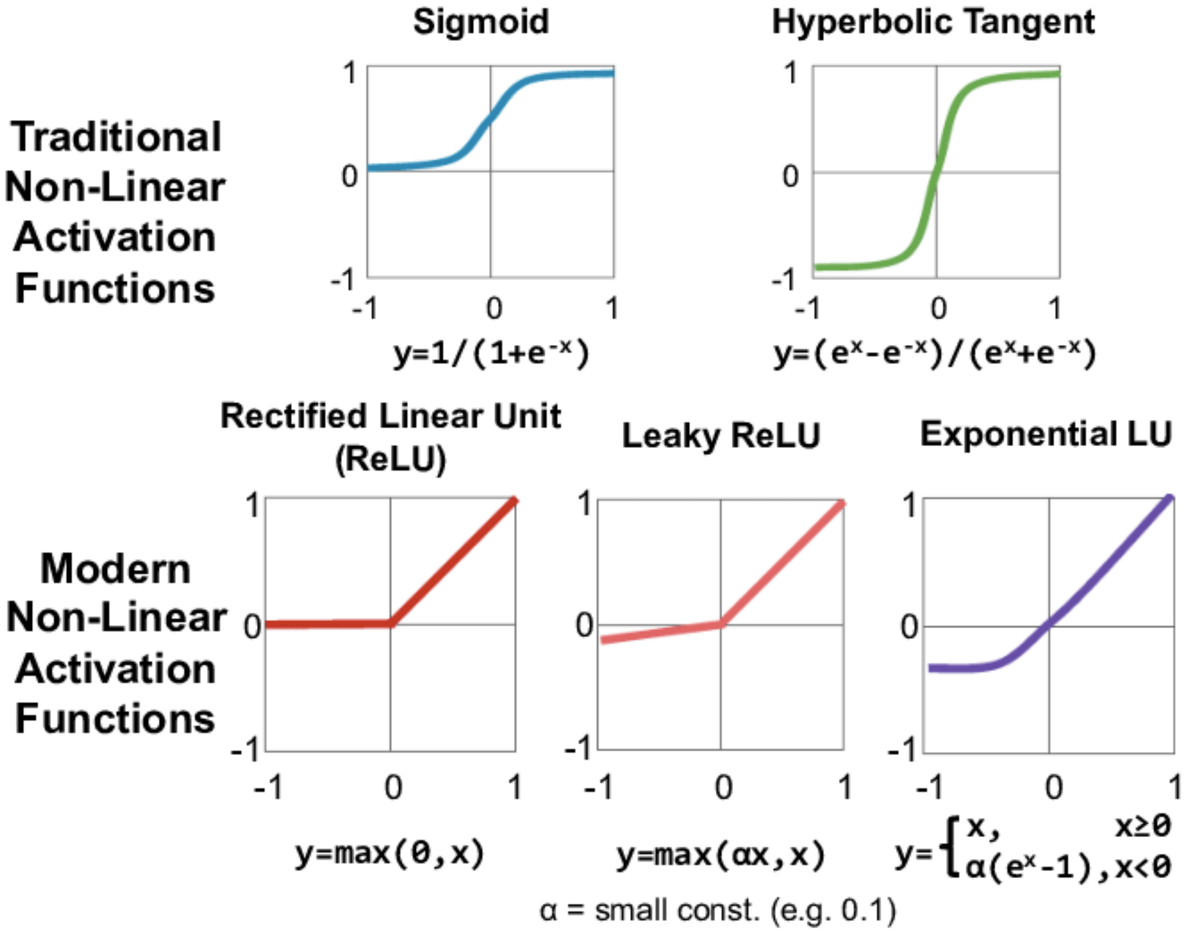
Input layer
(Layer 0)

Output layer
(Layer 1)

Activation Function (Sigmoid/Logistic)



Activation Functions



https://www.researchgate.net/publication/315667264_Efficient_Processing_of_Deep_Neural_Networks_A_Tutorial_and_Survey/figures?lo=1

Cost Function : Mean-Squared Errors (MSE)

A measure of how well the model fits the training data for the given model parameters.

Goal: Find best-fitted weights.

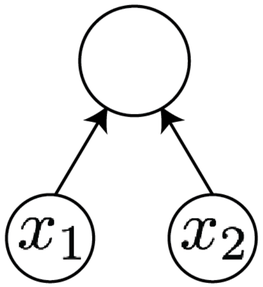
$$J(\boldsymbol{w}) = J(w_1, w_2) = \frac{1}{2N}\sum_{i=1}^{N} L(h(\boldsymbol{x}^{(i)}), y_i)$$

$$= \frac{1}{2N}\sum_{i=1}^{N} \text{Error}_i^2$$

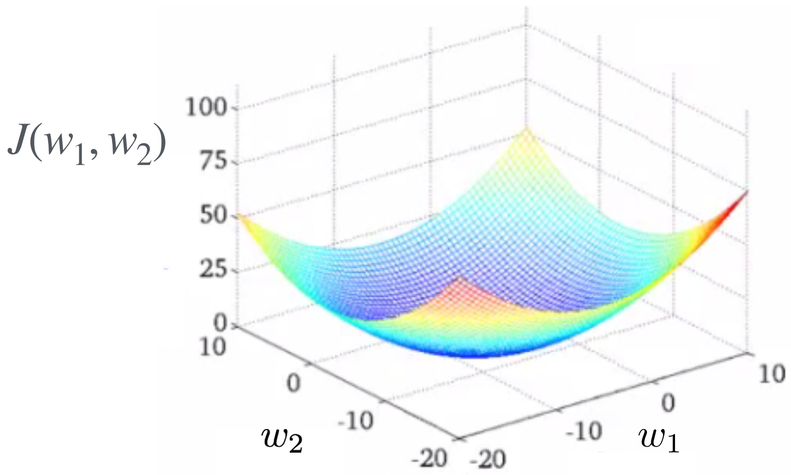
$$= \frac{1}{2N}\sum_{i} (h(\boldsymbol{x}^{(i)}) - y^{(i)})^2$$

$$= \frac{1}{2N}\sum_{i} ((w_1 x_1^{(i)} + w_2 x_2^{(i)} - y^{(i)})^2$$

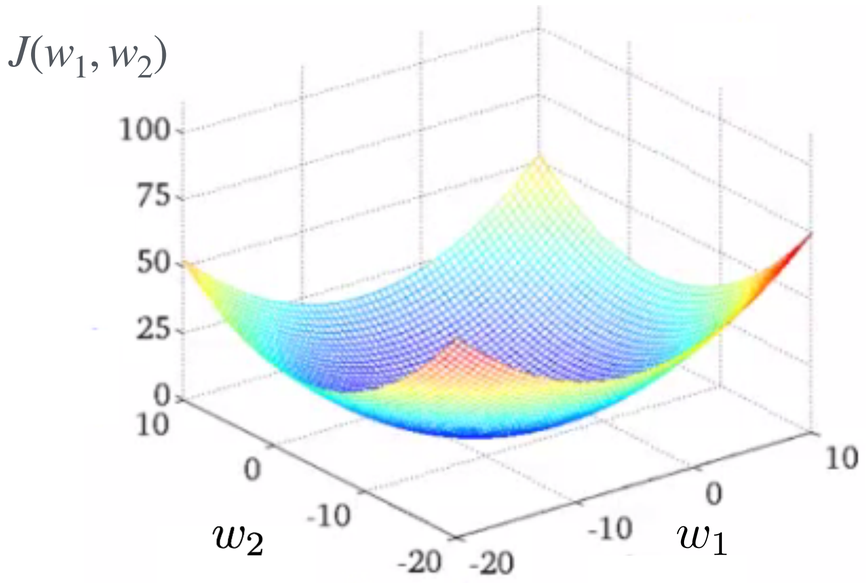
$$h = f(\Sigma_j w_j x_j)$$



Assume **linear activation function** and **Squared loss function**



Cost Function Landscape



- Where do "best" w_1 and w_2 locate in the loss function landscape ?
- Will the bowl look different if our training set $\{x, y\}$ are different?
- Starting at a random location on (w_1, w_2) plane, how do we move the point at the bottom of the bowl?

$$J(w_1, w_2) = \frac{1}{2N} \sum_i ((w_1 x_1^{(i)} + w_2 x_2^{(i)} - y^{(i)})^2$$

(Linear activation function)

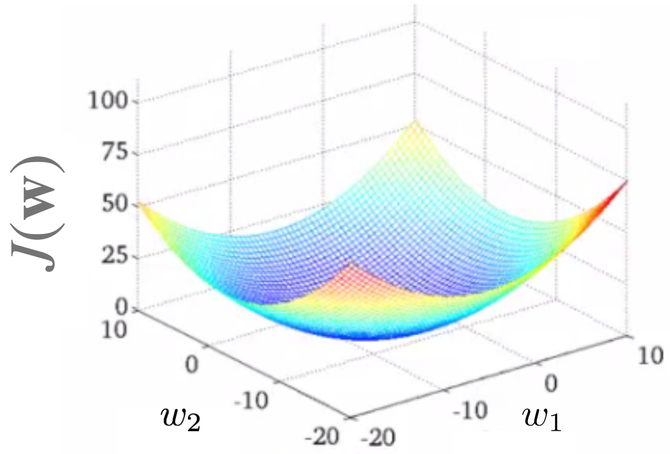
$$\frac{\partial J}{\partial w_1} = \frac{1}{N} \sum_i ((w_1 x_1^{(i)} + w_2 x_2^{(i)} - y^{(i)}) \cdot x_1^{(i)}$$

$$= \frac{1}{N} \sum_i \text{Error}_i \cdot x_1^{(i)}$$

All training instances are used to compute the gradient element.

$$\frac{\partial J}{\partial w_2} = \frac{1}{N} \sum_i \text{Error}_i \cdot x_2^{(i)}$$

Gradient vector: $\nabla J(\boldsymbol{w})$ or $\nabla J(w_1, w_2) \triangleq \frac{\partial J}{\partial \boldsymbol{w}} = \begin{bmatrix} \frac{\partial J}{\partial w_1} \\ \frac{\partial J}{\partial w_2} \end{bmatrix}$



Negative of gradient vector at (w_1, w_2) is the direction with largest decrease rate of $J(\boldsymbol{w})$, called the steepest descent direction.

Weight Adjustment: Gradient Descent Algorithm

Iteratively update the weight vector using

$$\boldsymbol{w}^{(t+1)} \Leftarrow \boldsymbol{w}^{(t)} + \Delta\boldsymbol{w}$$

$$\Leftarrow \boldsymbol{w}^{(t)} - \alpha \cdot \left.\frac{\partial J}{\partial \boldsymbol{w}}\right|_{\boldsymbol{w}=\boldsymbol{w}^{(t)}} = \boldsymbol{w}^{(t)} - \alpha \nabla J(\boldsymbol{w}^{(t)}), \quad \alpha > 0$$

Current weight

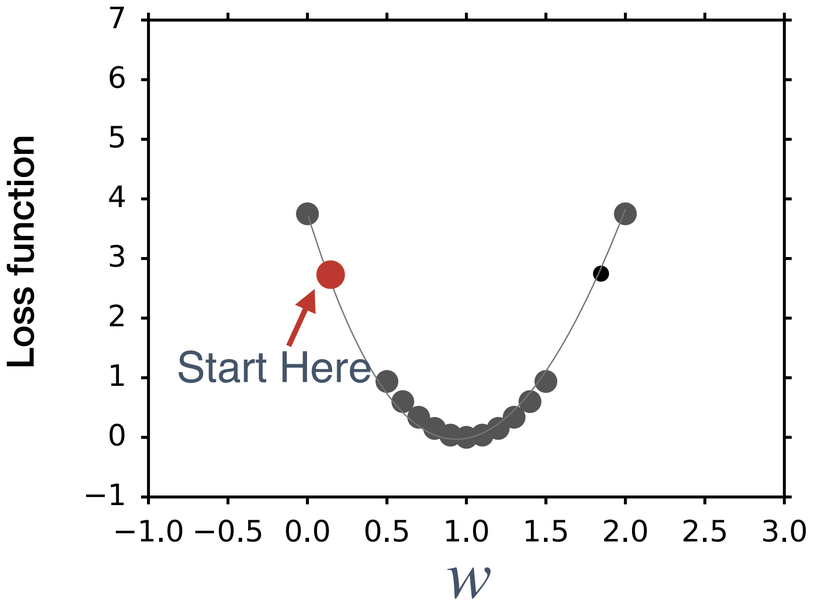
Learning rate

Gradient of cost function wrt w, i.e, Gradient vector

$$\begin{bmatrix} w_1^{(t+1)} \\ w_2^{(t+1)} \end{bmatrix} \Leftarrow \begin{bmatrix} w_1^{(t)} \\ w_2^{(t)} \end{bmatrix} - \alpha \cdot \left.\begin{bmatrix} \frac{\partial J}{\partial w_1} \\ \frac{\partial J}{\partial w_2} \end{bmatrix}\right|_{\boldsymbol{w}=\boldsymbol{w}^{(t)}}$$

$$\boldsymbol{w}^{(t+1)} \Leftarrow \boldsymbol{w}^{(t)} + \Delta \boldsymbol{w}$$

$$\Leftarrow \boldsymbol{w}^{(t)} - \alpha \cdot \left. \frac{\partial J}{\partial \boldsymbol{w}} \right|_{\boldsymbol{w}=\boldsymbol{w}^{(t)}}$$



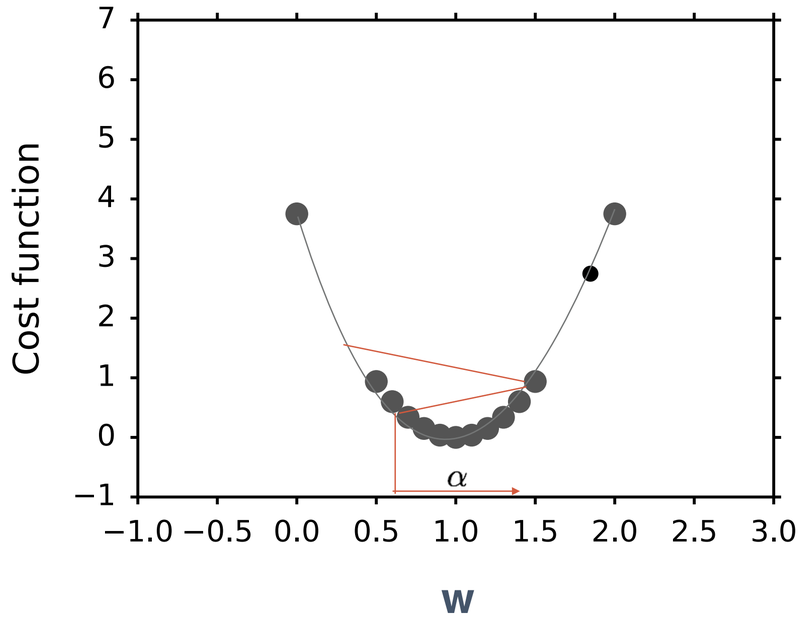
Consider the initial condition of weight w at the red point.

- Will the gradient be positive or negative?
- Will the algorithm suggests you to increase or decrease the value of w ?
- Will the algorithm walks to the right or the left?
- When should the algorithm stop ?

Picking Learning Rate α

$$\boldsymbol{w}^{(t+1)} \Leftarrow \boldsymbol{w}^{(t)} + \Delta \boldsymbol{w}$$

$$\Leftarrow \boldsymbol{w}^{(t)} - \alpha \cdot \frac{\partial J}{\partial \boldsymbol{w}}\bigg|_{\boldsymbol{w}=\boldsymbol{w}^{(t)}}$$



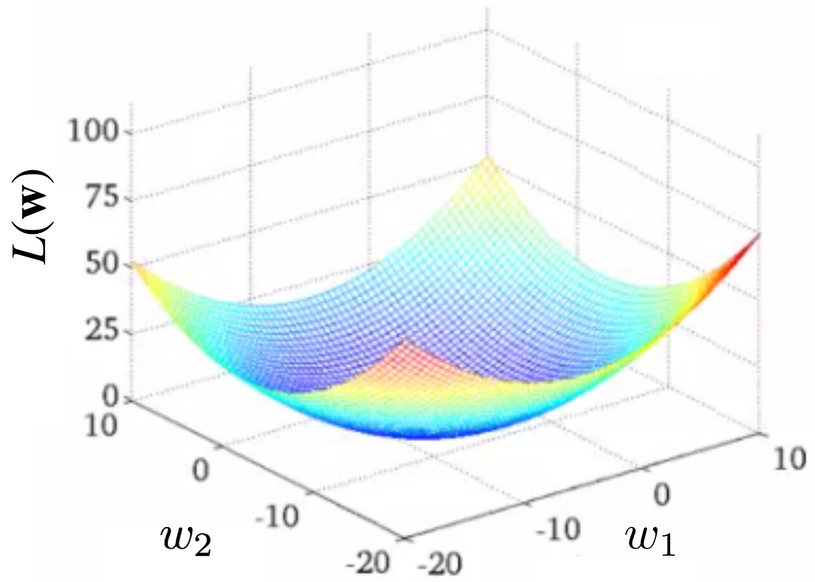
α is usually between 0 and 1

Large α: big step downhill

Small α: small step downhill

You don't want too big or too small α !

Summary of Gradient Descent Algorithm



- Take a step down the bowl with the length of the footstep α
- Each step, you will move from one point to another:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \nabla J(\boldsymbol{w})$$

$$\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \leftarrow \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} - \alpha * \nabla J(w_1, w_2)$$

Stopping Criteria: $\|\nabla J(w_1, w_2)\| < \varepsilon$

Sum of squared elements

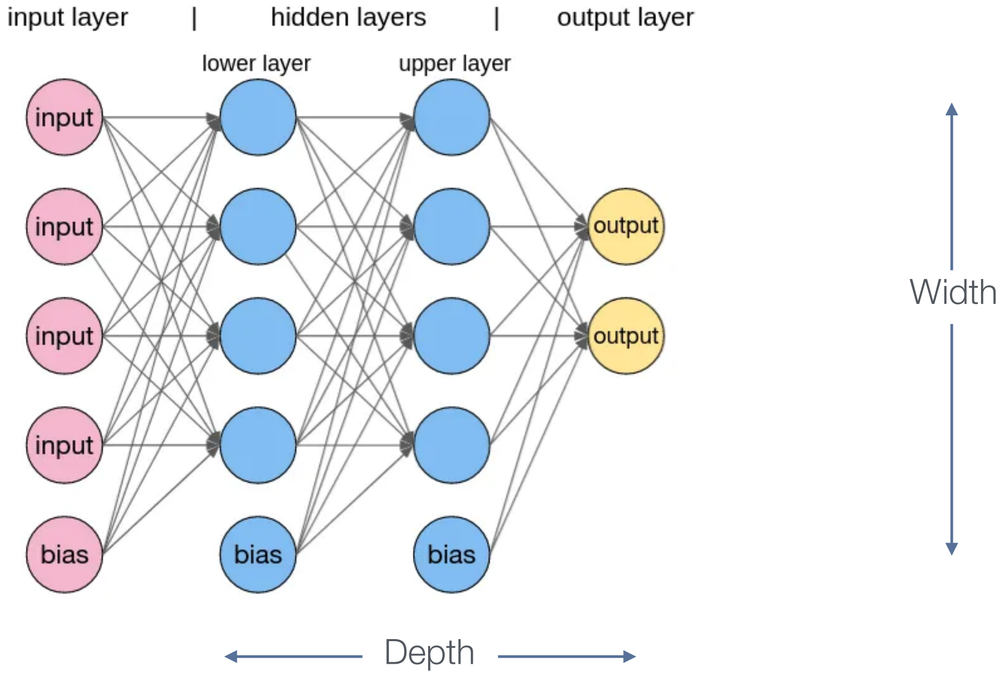
Tolerance

Neural Network (NN)

One or more hidden layers can be added between the input and output layer, aka a ***multi-layer perceptron***.

Called a ***feed-forward neural network*** if data only flows forward from the input layer.

Universal Approximation Theorem: NN with sufficient # hidden layers and units and non-linear activation functions can approximate any continuous 1D function to an arbitrary level of accuracy.

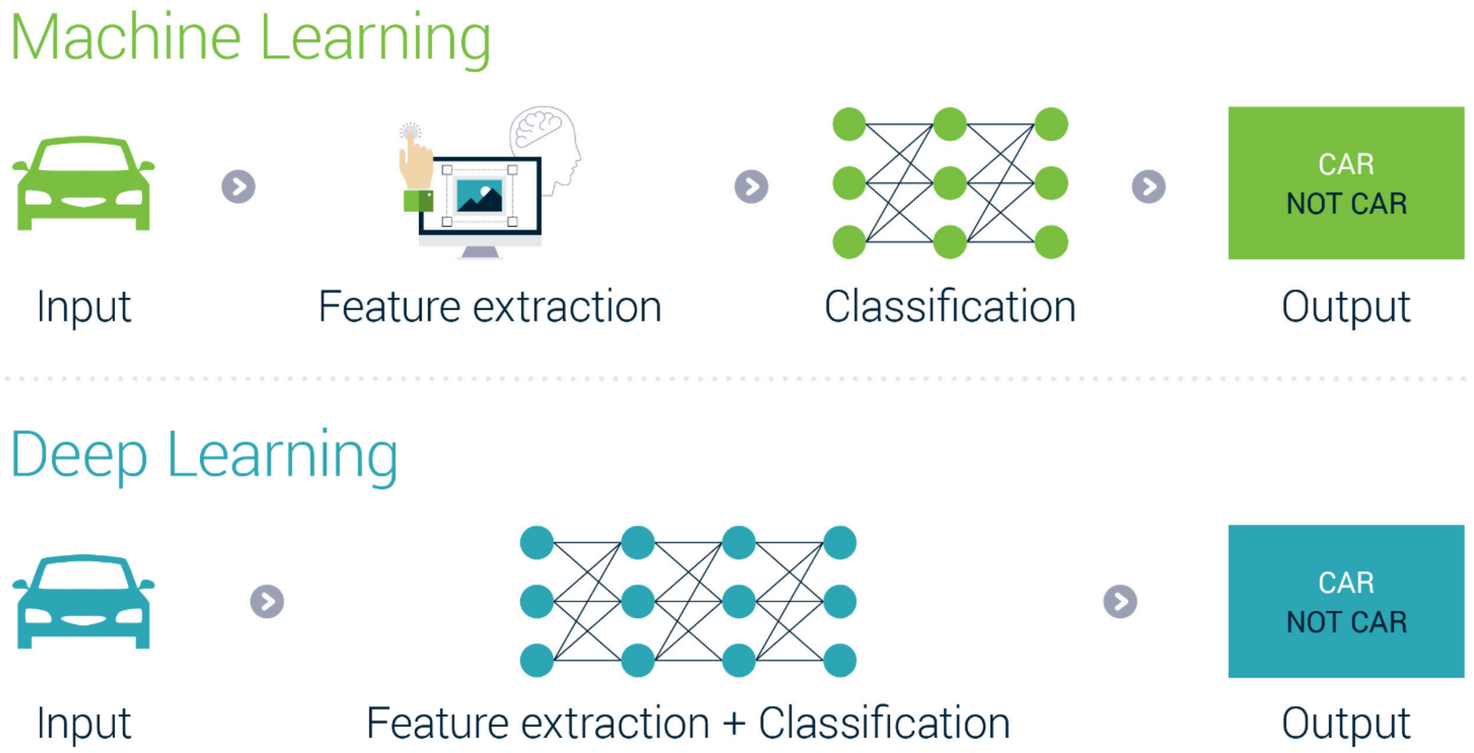


Introduction to how an Multilayer Perceptron works but without complicated math
https://medium.com/codex/introduction-to-how-an-multilayer-perceptron-works-but-without-complicated-math-a423979897ac

Deep Neural Network (DNN)

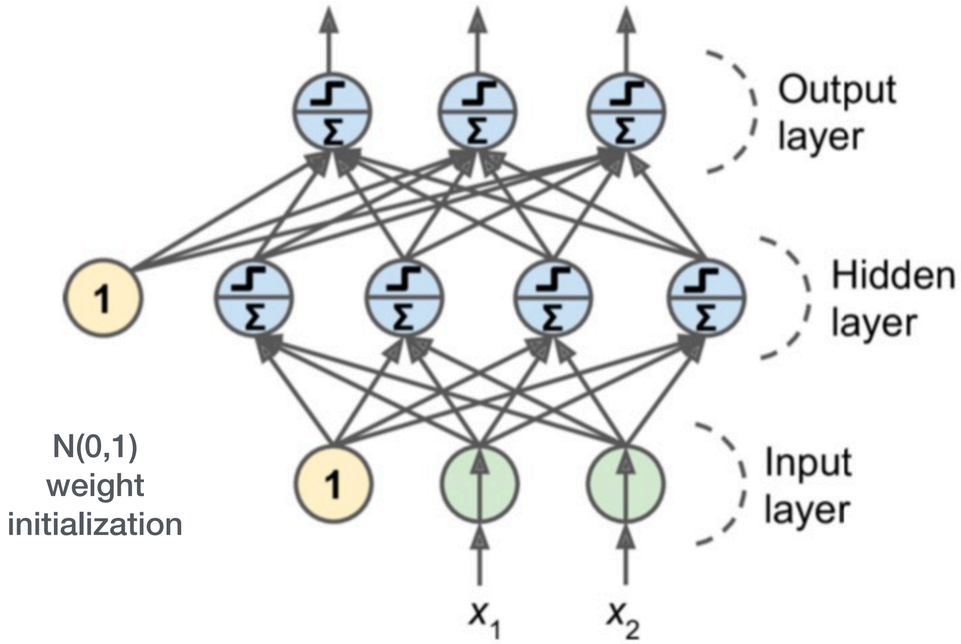
Neural networks with two or more hidden layers.

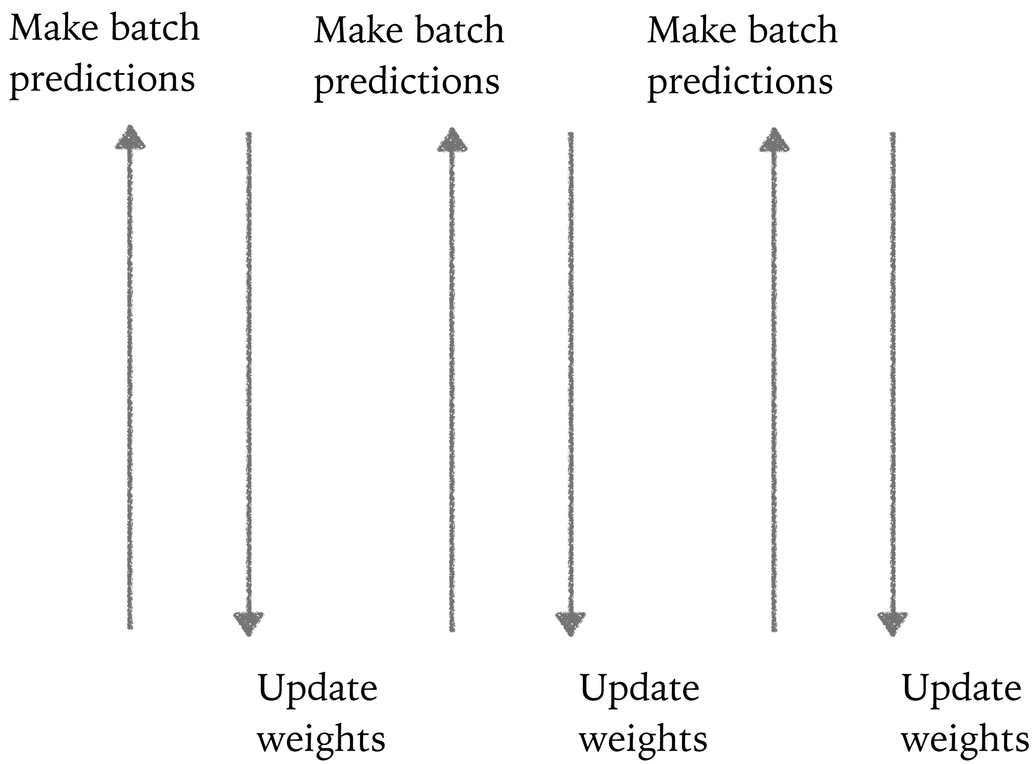
Called ***shallow neural network*** otherwise.



Source: verhaert.com

Training Feed-forward Networks: Backpropagation Gradient-Descent Algorithm





One *epoch* = All instances in the *train data set* have been evaluated.
Repeat several epochs as needed (usually with schedule of learning rate α)

Variants of Gradient Descent

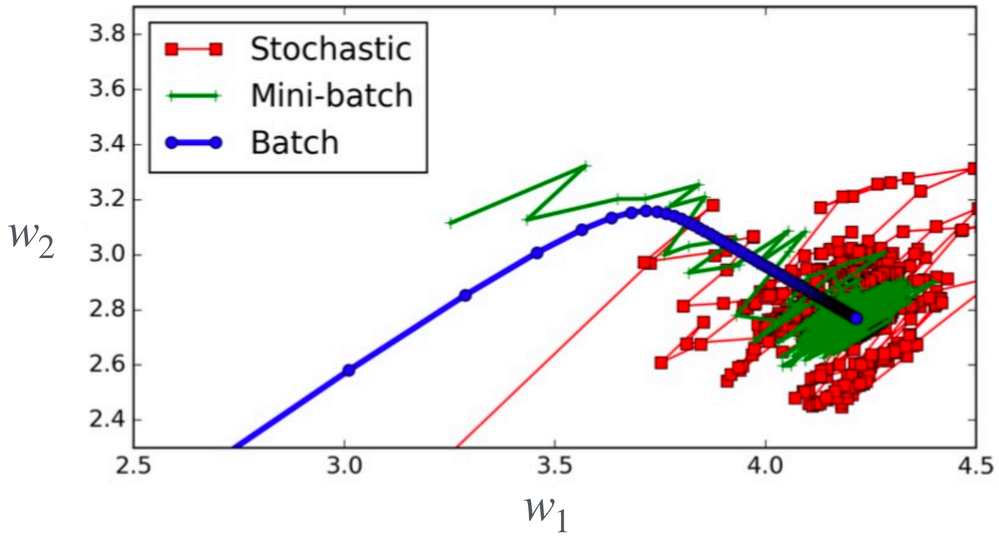
Batch GD

- Sum errors of all training instances/samples to compute the gradient vector.
- May converge slower as the whole training set is used per weight update.

Stochastic GD: Use one instance per weight update.

Mini-batch GD

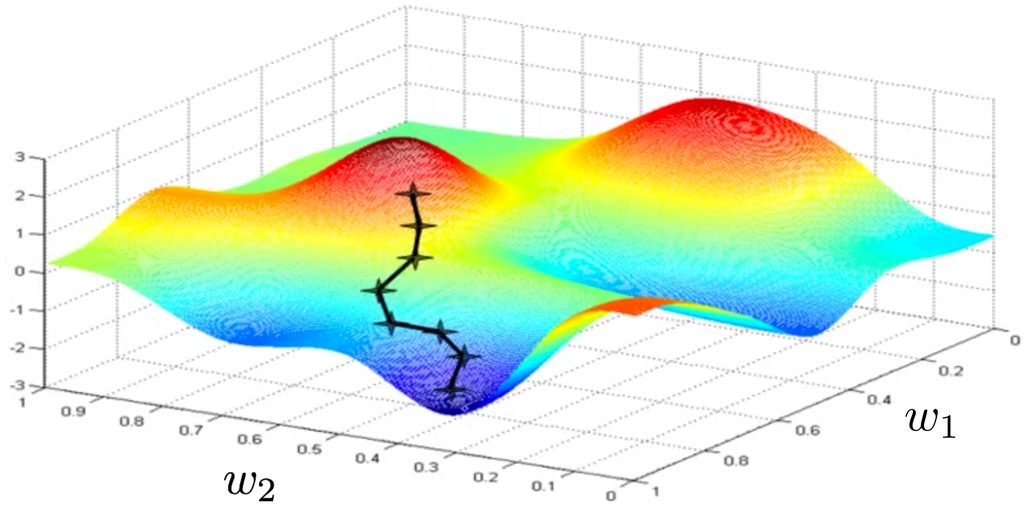
- Divide the whole data into batches.
- Use one batch to calculate the sum of errors per weight update.
- Popular batch sizes -- 16, 32, 64



Cost Function Landscape of (Deep) Neural Networks

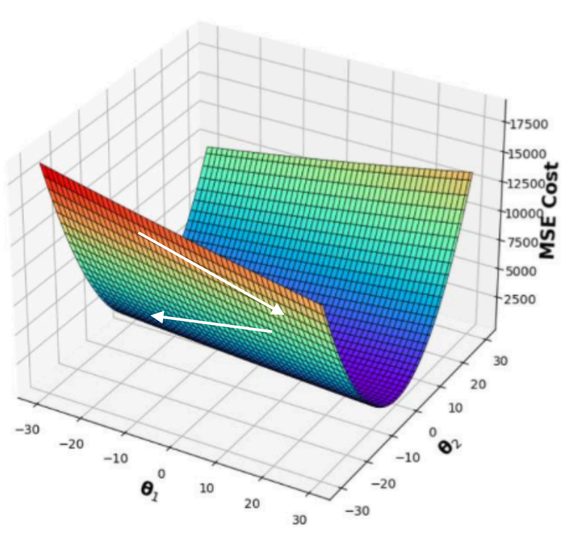
Generally have multiple local minima and plateaus

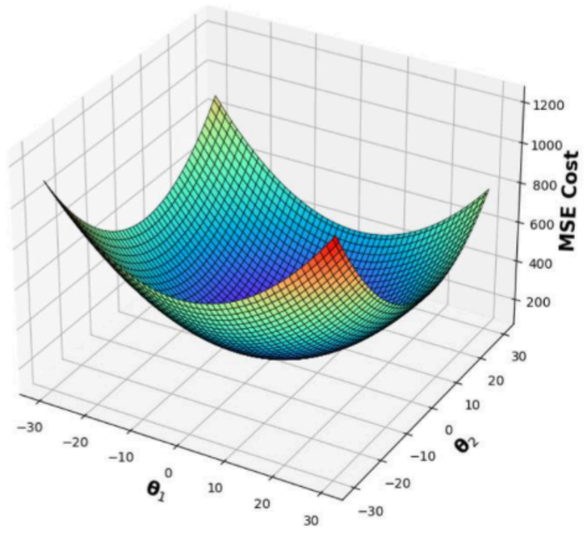
Solutions affected by initial weights and learning rate.



Effect of Data Scaling in Gradient Descent Algorithm

Elongated cost function
with unscaled data





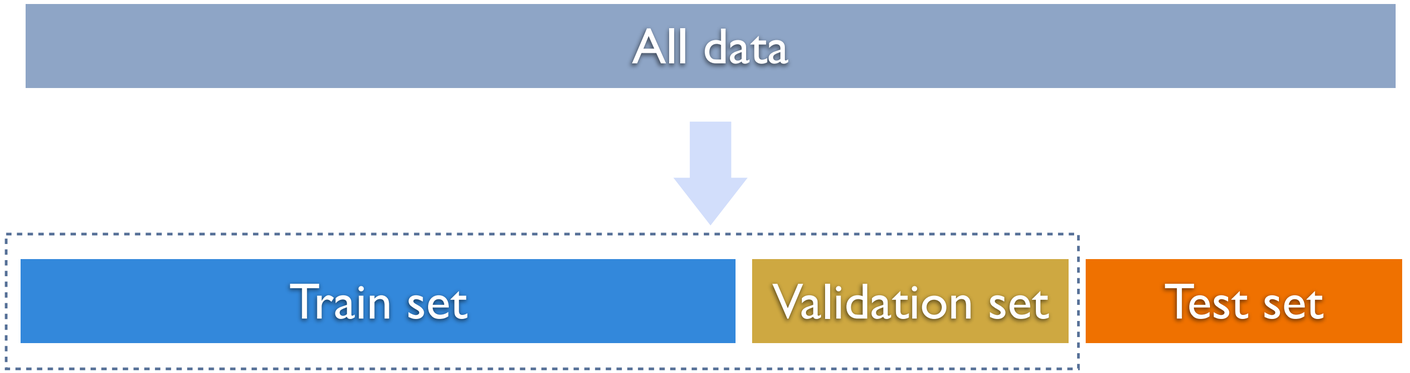
Model Assessment: Three-way Hold-out Method (Train-Validation-Test Split)

Train data: Used to fit the model over different hyperparameter settings or feature sets.

Validation data: Used to evaluate the models fitted by the train data.

Test data: Held out until the final hyperparameter setting and feature set have been decided.

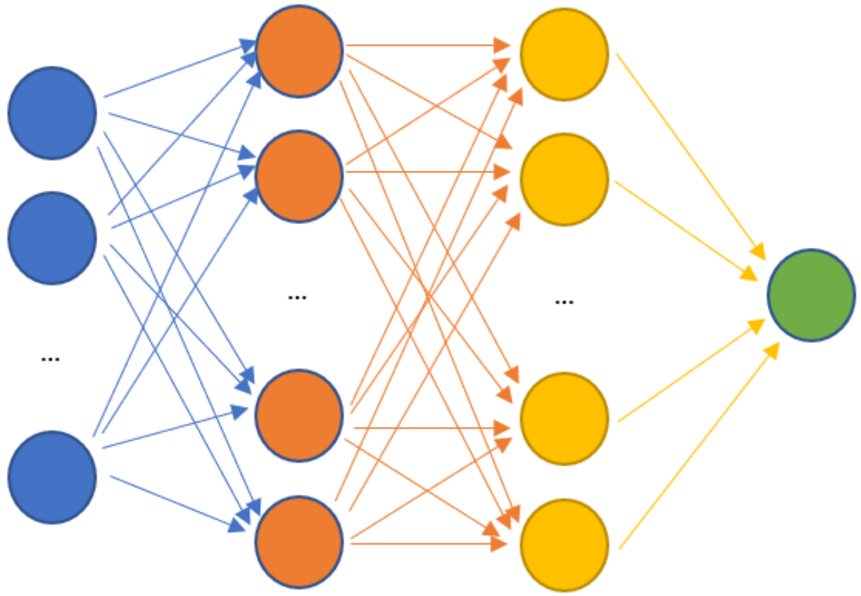
- Merge the train and validation data and fit the model.
- Evaluate for the generalization performance of the fitted model on the test set.



Use the whole data set to fit the model for deployment.

Neural Network Regression using Keras

Scaled inputs
(mandatory)



ReLu $(y > 0)$
Linear $(-\infty < y < \infty)$
Logistic $(0 < y < 1)$
Tanh $(-1 < y < -1)$

Basic NN hyperparameters

Network architecture/configuration
Learning rate
Optimizer algorithm
Batch size
Hidden layers
Activation function

Run `conda install keras tensorflow` to install keras in Anaconda.

See Jupyter Notebook

```python
import pandas as pd

house_df = pd.read_csv('data/house_sales.csv', sep='\t')
house_df.head()
```

	ID	DocumentDate	SalePrice	PropertyID	PropertyType	ym	zhvi_px	zhvi_idx	AdjSalePrice	NbrLivingUnits	...	Bathrooms	Bedrooms	BldgGrade
0	1	2014-09-16	280000	1000102	Multiplex	2014-09-01	405100	0.930836	300805.0	2	...	3.00	6	7
1	2	2006-06-16	1000000	1200013	Single Family	2006-06-01	404400	0.929228	1076162.0	1	...	3.75	4	10
2	3	2007-01-29	745000	1200019	Single Family	2007-01-01	425600	0.977941	761805.0	1	...	1.75	4	8
3	4	2008-02-25	425000	2800016	Single Family	2008-02-01	418400	0.961397	442065.0	1	...	3.75	5	7
4	5	2013-03-29	240000	2800024	Single Family	2013-03-01	351600	0.807904	297065.0	1	...	1.75	4	7

5 rows × 23 columns

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22689 entries, 0 to 22688
Data columns (total 23 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   ID               22689 non-null  int64
 1   DocumentDate     22689 non-null  object
 2   SalePrice        22689 non-null  int64
 3   PropertyID       22689 non-null  int64
 4   PropertyType     22689 non-null  object
 5   ym               22689 non-null  object
 6   zhvi_px          22689 non-null  int64
 7   zhvi_idx         22689 non-null  float64
 8   AdjSalePrice     22689 non-null  float64
 9   NbrLivingUnits   22689 non-null  int64
 10  SqFtLot          22689 non-null  int64
 11  SqFtTotLiving    22689 non-null  int64
 12  SqFtFinBasement  22689 non-null  int64
 13  Bathrooms        22689 non-null  float64
 14  Bedrooms         22689 non-null  int64
 15  BldgGrade        22689 non-null  int64
 16  YrBuilt          22689 non-null  int64
 17  YrRenovated      22689 non-null  int64
 18  TrafficNoise     22689 non-null  int64
 19  LandVal          22689 non-null  int64
 20  ImpsVal          22689 non-null  int64
 21  ZipCode          22689 non-null  int64
 22  NewConstruction  22689 non-null  bool
dtypes: bool(1), float64(3), int64(16), object(3)
```

```
num_columns = ["SqFtTotLiving", "SqFtLot", "Bathrooms", "Bedrooms",
               "BldgGrade", "NbrLivingUnits","SqFtFinBasement"]
cat_columns = ['PropertyType', 'NewConstruction']
target_column = ['AdjSalePrice']
house_df = house_df[num_columns + cat_columns + target_column]
```

```
from sklearn.model_selection import train_test_split

house_train, house_test = train_test_split(house_df, test_size=0.25, random_state=0, shuffle=True)
```

```python
# Create transformation pipeline
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer

# Can add more steps to the pipeline
num_pipeline = Pipeline([
    ('std_scaler', StandardScaler())
])


full_pipeline = ColumnTransformer([
    ("num", num_pipeline, num_columns),
    ("cat", OneHotEncoder(drop='first', handle_unknown='ignore',
                          dtype=int, sparse_output=False), cat_columns)
], remainder='drop')
```

```python
# Apply the pipeline to features
X_transformed = full_pipeline.fit_transform(house_train)

transformed_feature_names = full_pipeline.get_feature_names_out()
X_train = pd.DataFrame(X_transformed, columns=transformed_feature_names)
y_train = np.log(house_train[["AdjSalePrice"]]).reset_index(drop=True)
```

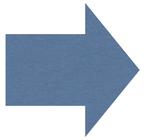
	num__SqFtTotLiving	num__SqFtLot	num__Bathrooms	num__Bedrooms	num__BldgGrade	num__NbrLivingUnits	num__SqFtFinBas
0	-0.684508	-0.359271	0.093487	-1.497623	-0.579029	-0.121897	-0.6
1	-1.180855	-0.123994	-0.557272	-1.497623	-0.579029	-0.121897	-0.6
2	0.650115	-0.171969	0.418867	0.690746	1.122508	-0.121897	-0.6
3	-0.552149	-0.111079	-0.882652	0.690746	-0.579029	-0.121897	-0.6
4	-0.166101	0.465743	0.418867	-0.403439	1.122508	-0.121897	-0.6

```
0    12.790272
1    12.869268
2    13.508154
3    12.678897
4    13.336490
```

```python
print(f'Train set Feature shape: {X_train.shape}, Dim: {X_train.ndim}')
print(f'Train set Target shape: {y_train.shape}, Dim: {y_train.ndim}')
```

```
Train set Feature shape: (17016, 10), Dim: 2
Train set Target shape: (17016, 1), Dim: 2
```

```
# ### Create a fully-connected neural network

from keras.layers import Dense
from keras.models import Sequential

network = Sequential([
  Dense(16, activation='tanh', input_shape=X_train.shape[1:]),
  Dense(4, activation='tanh'),
  Dense(1, activation='relu')
])

# Compile the model by specifying the hyperparameter and loss function to determine the weights.
network.compile(optimizer='adam', loss='mse', metrics=['mse'])
network.summary()
```



The network has two hidden layers, each with 16 and 4 nodes of tanh activation, which will be denoted by **tanh(16, 4)** hidden layer configuration.

Model: "sequential_48"

Layer (type)	Output Shape	Param #
dense_195 (Dense)	(None, 16)	176
dense_196 (Dense)	(None, 4)	68
dense_197 (Dense)	(None, 1)	5

Total params: 249 (996.00 B)

Trainable params: 249 (996.00 B)

Non-trainable params: 0 (0.00 B)

```python
# Fit the model
history = network.fit(X_train, y_train,
                      epochs=50,
                      batch_size=16,
                      verbose=1,
                      validation_split=0.2)
```

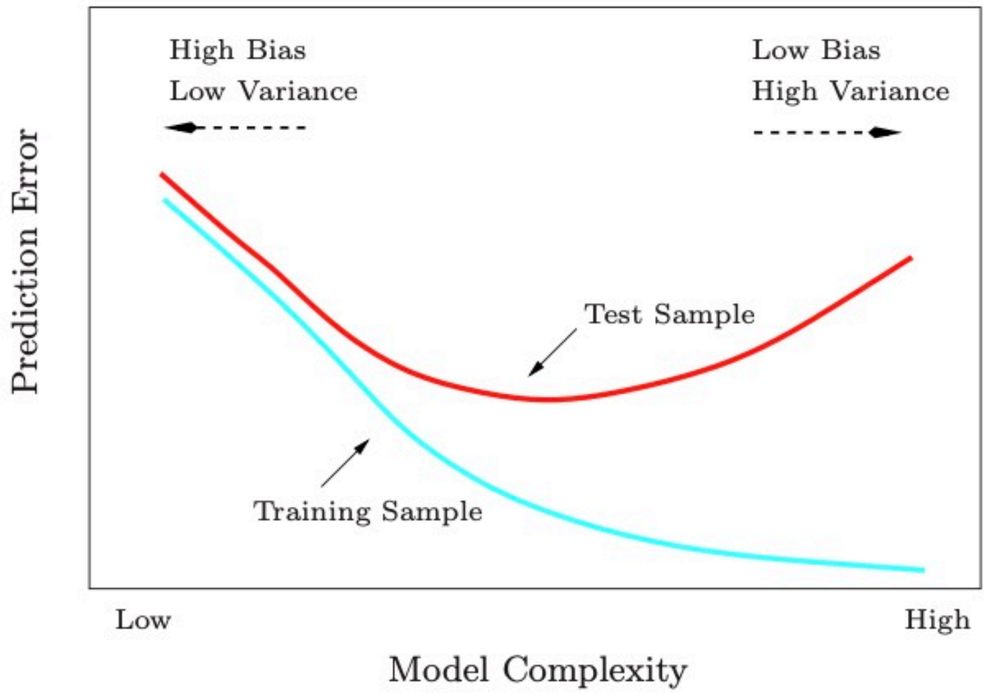
```
Train on 13612 samples, validate on 3404 samples
Epoch 1/50
13612/13612 [==============================] - 3s 243us/step - loss: 172.2081 - val_loss: 172.1556
Epoch 2/50
13612/13612 [==============================] - 2s 172us/step - loss: 93.7316 - val_loss: 0.5385
Epoch 3/50
13612/13612 [==============================] - 2s 166us/step - loss: 0.2958 - val_loss: 0.2536
Epoch 4/50
13612/13612 [==============================] - 2s 164us/step - loss: 0.2194 - val_loss: 0.2288
Epoch 5/50
13612/13612 [==============================] - 2s 180us/step - loss: 0.2005 - val_loss: 0.2098
Epoch 6/50
13612/13612 [==============================] - 3s 207us/step - loss: 0.1849 - val_loss: 0.1915

..

Epoch 48/50
13612/13612 [==============================] - 3s 208us/step - loss: 0.0906 - val_loss: 0.0974
Epoch 49/50
13612/13612 [==============================] - 3s 220us/step - loss: 0.0900 - val_loss: 0.0985
Epoch 50/50
13612/13612 [==============================] - 3s 195us/step - loss: 0.0905 - val_loss: 0.1010
```

```python
# Save the transformation pipeline and the fitted model
pickle.dump(full_pipeline, open("house_pipeline.pickle", "wb"))
pickle.dump(network, open("house_nn_model.pickle", "wb"))
```

```python
train_loss, train_metric = network.evaluate(X_train, y_train)
print(f'Train Loss: {train_loss}\nTrain Metric: {train_metric}')
```

```
532/532 [==============================] - 0s 285us/step - loss: 0.0859 - wape: 1.6467
Train Loss: 0.08594581484794617
Train Metric: 1.646695613861084
```

Model Overfitting



Underfitting

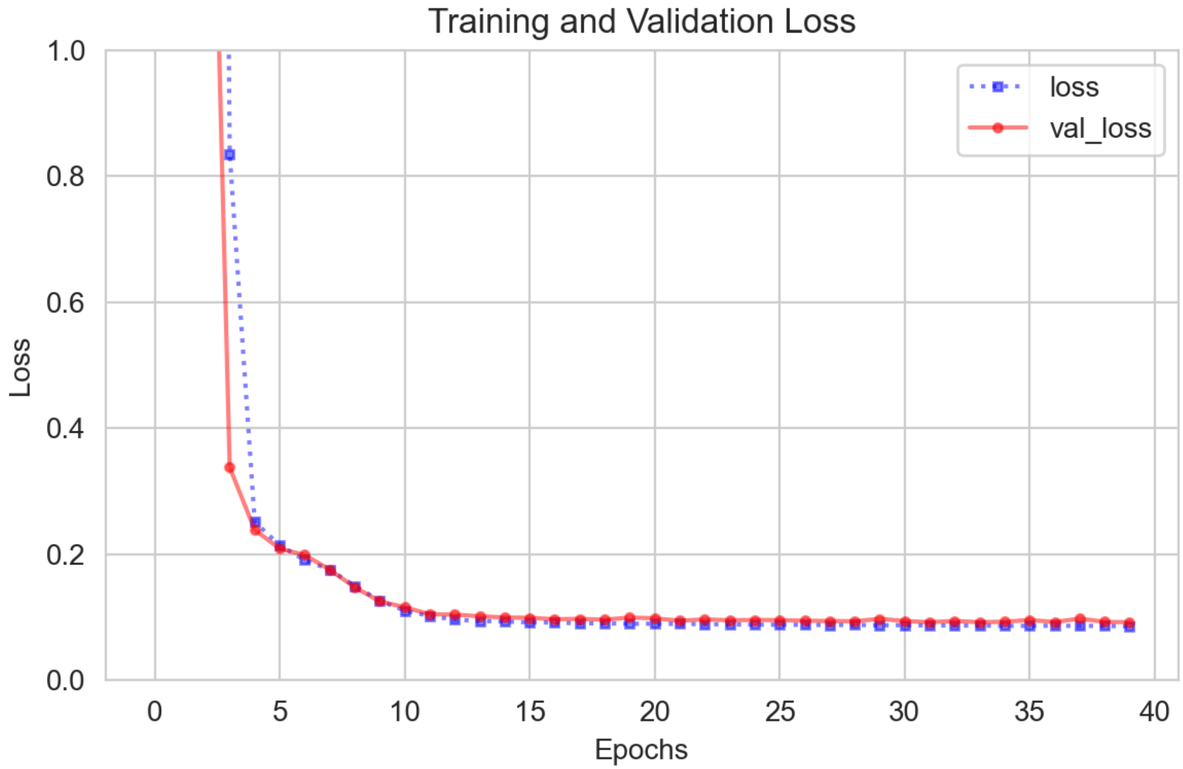
Overfitting

Learning Curve

```
history_df = pd.DataFrame(history.history)
history_df
```

	loss	wape	val_loss	val_wape
0	89.301285	70.231422	41.619595	49.054611
1	23.713831	36.283421	11.052995	25.079287
2	5.407795	16.758427	1.900781	9.846708
3	0.835170	5.785215	0.337909	3.284475
4	0.251808	2.843040	0.238935	2.784113
5	0.214319	2.658514	0.209076	2.582684
6	0.191027	2.505762	0.199158	2.533142
7	0.175722	2.391473	0.175731	2.338510
8	0.150028	2.180242	0.147929	2.108515
9	0.126615	1.996229	0.125611	1.935427
10	0.110866	1.864465	0.116171	1.861445
11	0.102343	1.794024	0.105136	1.773318
12	0.096597	1.750457	0.104535	1.790033

```
history_df[['loss','val_loss']].plot(style=['bs:','ro-'],
                                     ms=3, lw=1.5, alpha=0.5, figsize=(6, 4));
plt.grid(True);
plt.gca().set_ylim(0, 1)
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.tight_layout();
```



Evaluate the train and test performance

```python
from sklearn.metrics import mean_absolute_percentage_error as MAPE
from sklearn.metrics import mean_squared_error as MSE

# Train performance on MSE and MAPE
y_pred = network.predict(X_train)

print(f'MSE on train data: {MSE(y_train,y_pred):2f}')
print(f'MAPE on train data: {MAPE(y_train,y_pred):.3f}%')
```

```
MSE on train data: 0.084379
MAPE on train data: 0.016%
```

```python
# Load the pipeline and fitted model and apply them to test data.
loaded_pipeline = pickle.load(open("house_pipeline.pickle", "rb"))
loaded_model = pickle.load(open("house_nn_model.pickle", "rb"))


X_test_transformed = loaded_pipeline.transform(house_test)
y_test = np.log(house_test[["AdjSalePrice"]])

# Test performance on MSE and MAPE
y_test_pred = loaded_model.predict(X_test_transformed)
print(f'MSE on test data: {MSE(y_test, y_test_pred):2f}')
print(f'MAPE on test data: {MAPE(y_test, y_test_pred):.3f}%')
```

```
MSE on test data: 0.085497
MAPE on test data: 0.017%
```

Your Turn: House Price Prediction

From the California housing dataset, develop a neural network regression model to predict the median house value by using only numeric features in the dataset. Train the model with 80% of data with the following network configurations of the hidden layers:

- tanh(20, 20, 10, 5)
- relu(20, 20, 10, 5)
- tanh(8, 5)
- relu(8, 5)

Examine the learning curve and the model train/test performance to assess the model quality.

```
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object
```

Load data from 'data/california-housing.csv'

Loss Function for Classification Problem: Cross-Entropy

For binary classification, the model predicts $0 < h(x_i) < 1$.

A loss value of a sample i can be defined by the ***Binomial log loss*** as

$$\text{Error}_i \text{ or } L_i(\boldsymbol{w}) = -(y_i \log_2(h(x_i;\boldsymbol{w})) + (1-y_i)\log_2(1-h(x_i;\boldsymbol{w})))$$

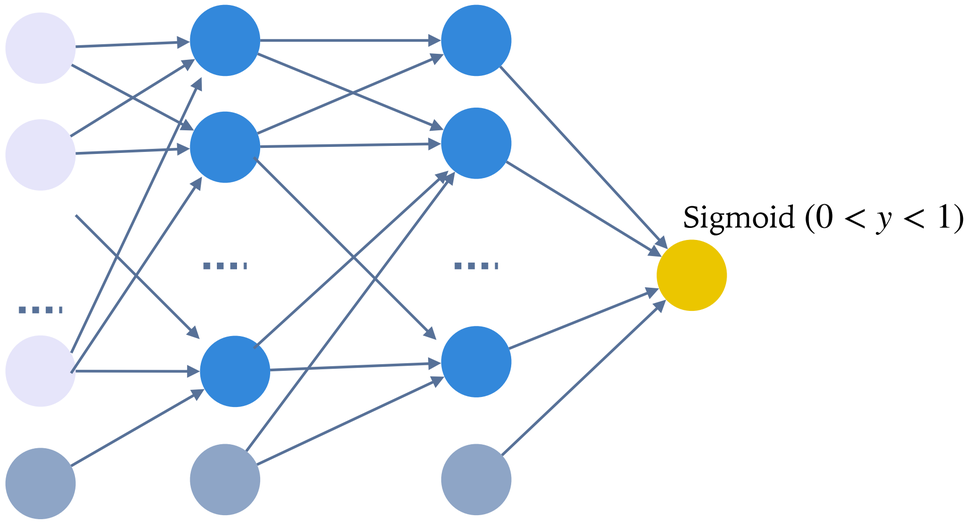
Actual value

Predict prob.

h	y	Squared erroor	Cross-entropy
0.8	1	0.04	0.10
0.1	1	0.81	1.00
0.1	0	0.01	0.05

$$J(\boldsymbol{w}) = \sum_{i=1}^{N} L_i(\boldsymbol{w})$$

Binary Classification



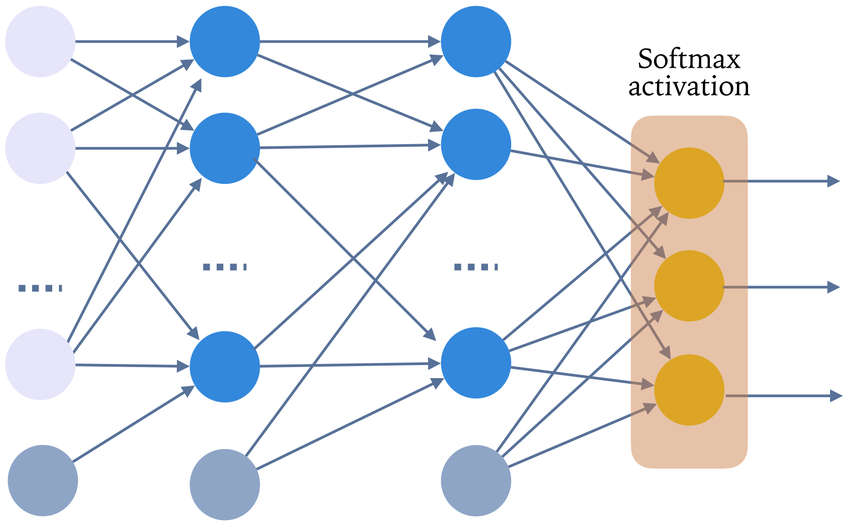
Input layer Hidden layers Output layer

```
network = Sequential([
  Dense(16, activation='tanh', input_shape=X_train.shape[1:]),
  Dense(4, activation='tanh'),
  Dense(1, activation='sigmoid')
])

network.compile(loss="binary_crossentropy",
                optimizer='adam')
```

```
y_pred = (network.predict(X_train) > 0.5).astype("int")
```

In Keras, `predict()` returns values in (0, 1) or probabilites.

Multiclass Classification



$$f(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}}$$

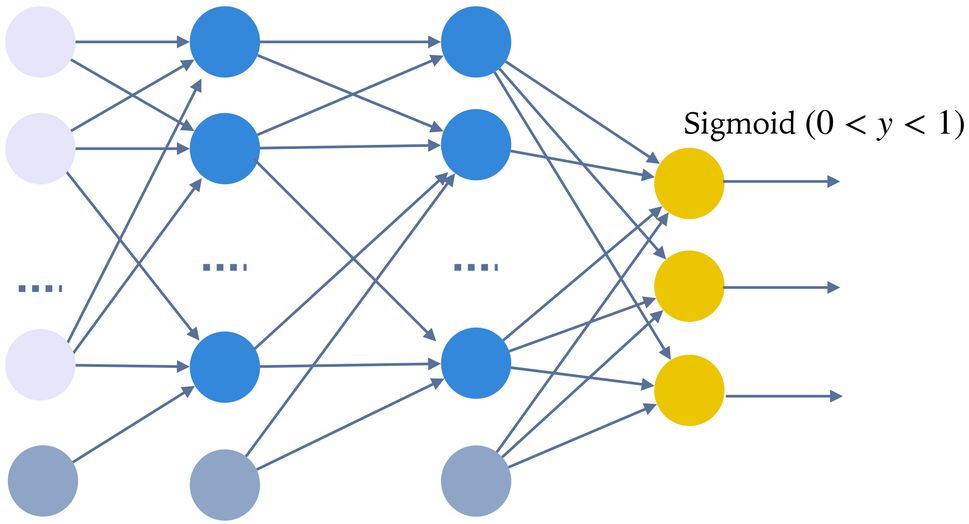
Input layer Hidden layers Softmax layer

```
NUMCLASS = 3

network = Sequential([
  Dense(16, activation='tanh',
            input_shape=X_train.shape[1:]),
  Dense(4, activation='tanh'),
  Dense(NUMCLASS, activation='softmax')
])

network.compile(loss="categorical_crossentropy",
                optimizer='adam')
```

```
y_pred = np.argmax(network.predict(X_train), axis=-1)
```

Multi-label Binary Classification

Unique to neural networks



Input layer Hidden layers Output layer

```
network = Sequential([
  Dense(16, activation='tanh', input_shape=X_train.shape[1:]),
  Dense(4, activation='tanh'),
  Dense(y_train.shape[1], activation='sigmoid')
])

network.compile(loss="binary_crossentropy",
                optimizer='adam')
```

```
y_pred = (network.predict(X_train) > 0.5).astype("int")
```

Fine-Tuning Deep Neural Networks

Unstable (vanishing and exploding) gradient problems

- Weight initialization
- Non-saturated activation function
- Batch normalization

Speed-up training time

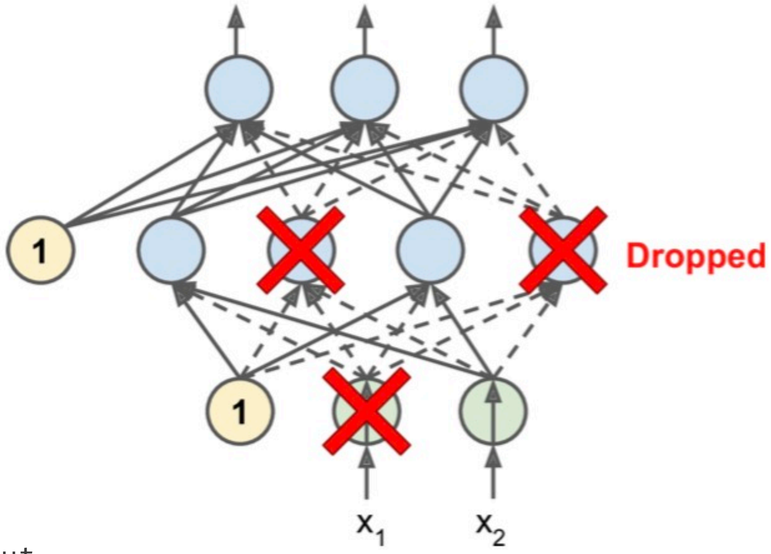
- Faster optimizers (momentum term)
- Pre-trained layers (Transfer learning)

Avoiding overfitting

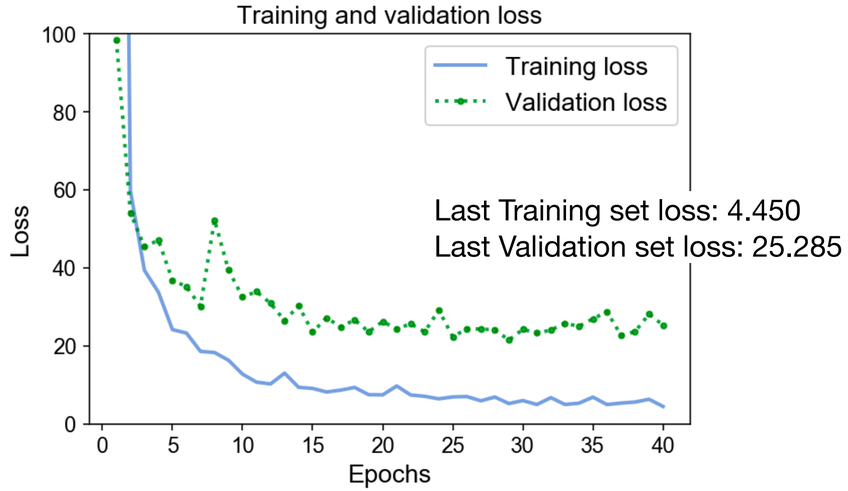
- Reduce # hidden layers
- Early Stopping
- Weight regularization
- Dropout layers

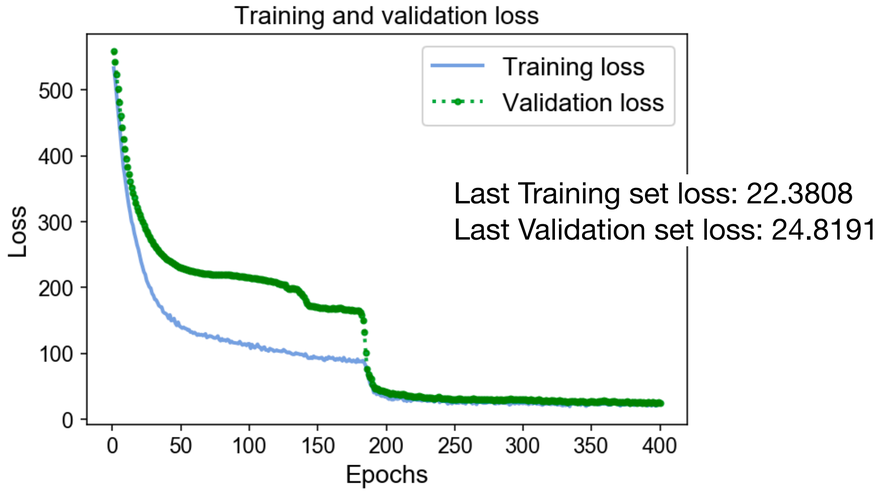
Avoiding Overfitting – Dropout Layer

Neurons in a dropout layer is ignored with a specific probability at each training step.




```
from keras.layers import Dropout

reg_network = Sequential([
  Dense(16, activation='tanh', input_shape=(X_train.shape[1]),
  Dropout(rate=0.2),
  Dense(8, activation='relu'),
  Dropout(rate=0.2),
  Dense(1, activation='relu')
])
```





Summary

Neural network capable of modeling complex relationship with high model capacity

Training and tuning can be difficult.

- Lots of hyperparameters to tune
- Flexible choices of network configurations
- Need relatively large amount of training data
- Vanishing/exploding gradients
- Computational intensive
- Prone to overfitting due to its high model capability.

Blackbox modeling, significance of variables not obvious.

Learning curve diagnosis helps identifying overfitting/underfitting.

Several techniques for model building - Weight initialization, Dropout, Regularization, etc.