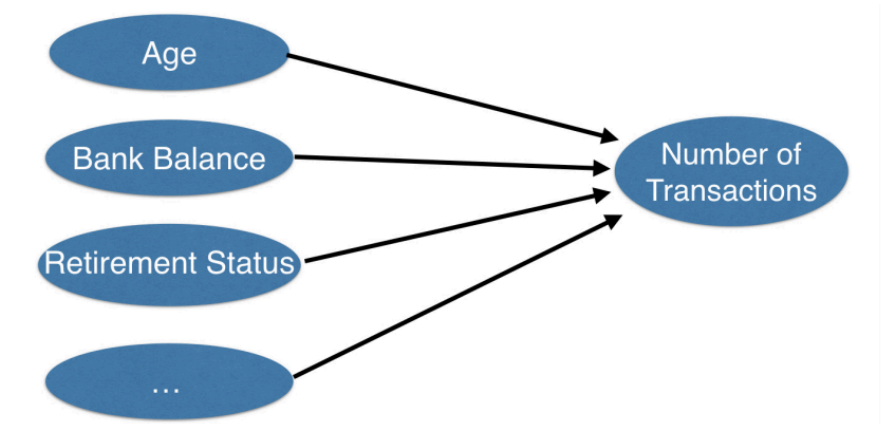
Deep Learning
Artificial Neural Network

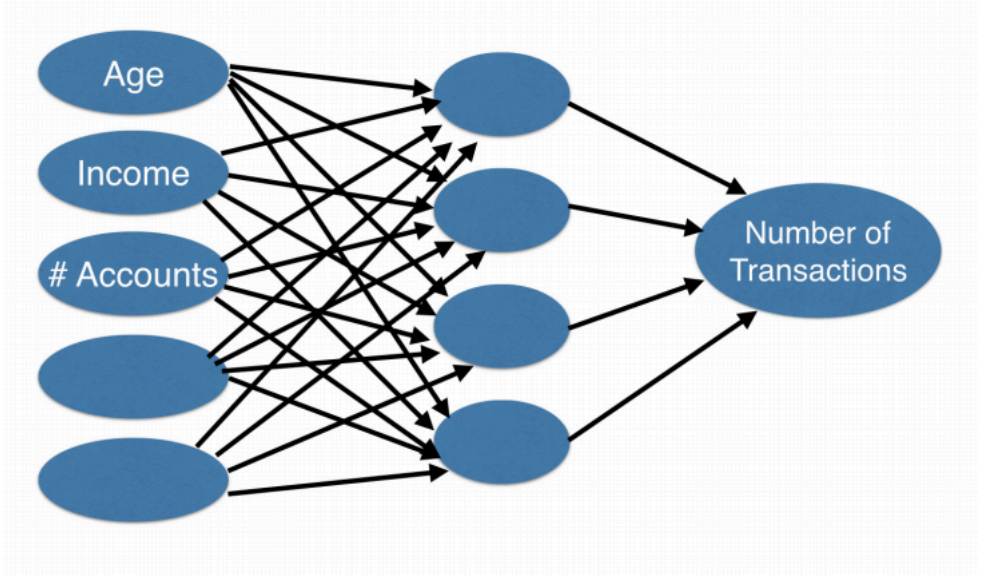
Classical regression vs.. Neural Network

- Imagine you are a data scientist for a bank
  - You need to predict how many transactions each customer will make next year

Linear Regression



Neural Network



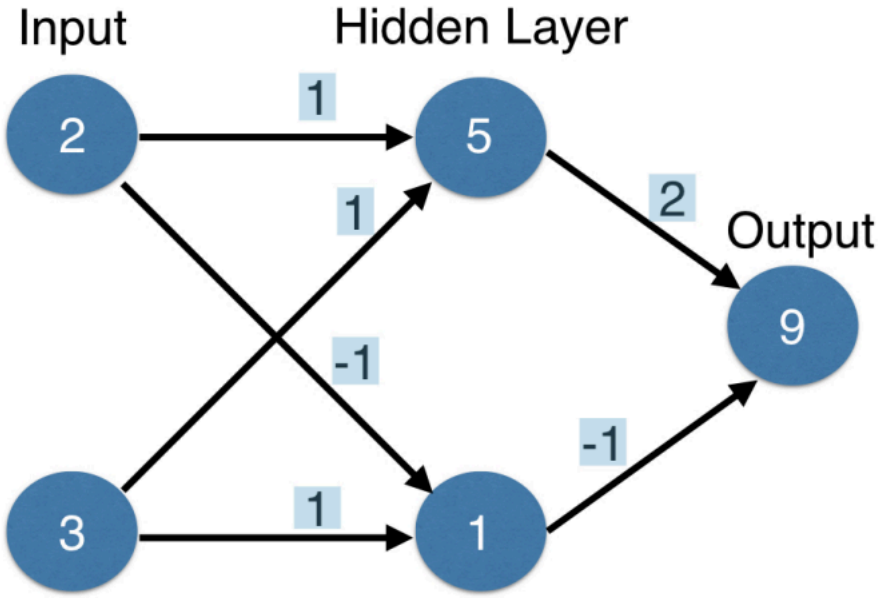
Interactions

- Neural networks account for interactions really well
- Deep learning uses especially powerful neural networks
  - Text
  - Images
  - Videos
  - Audio
  - Source code

Forward Propagation

Bank transactions example

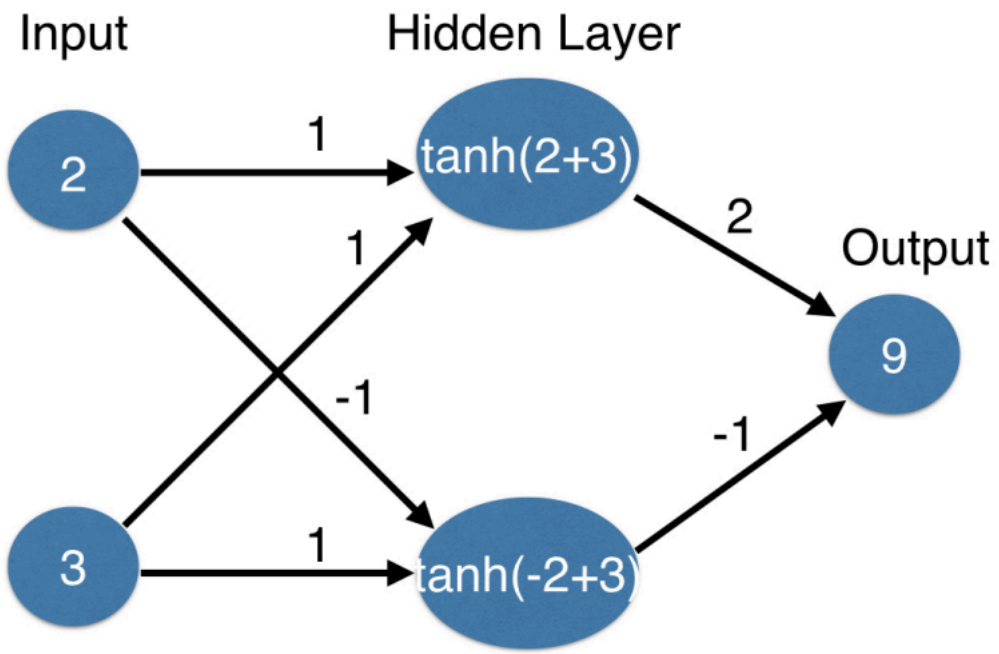
- Make predictions based on:
  - Number of children
  - Number of existing accounts



- Multiply - add process
- Dot product
- Forward propagation for one data point at a time
- Output is the prediction for that data point

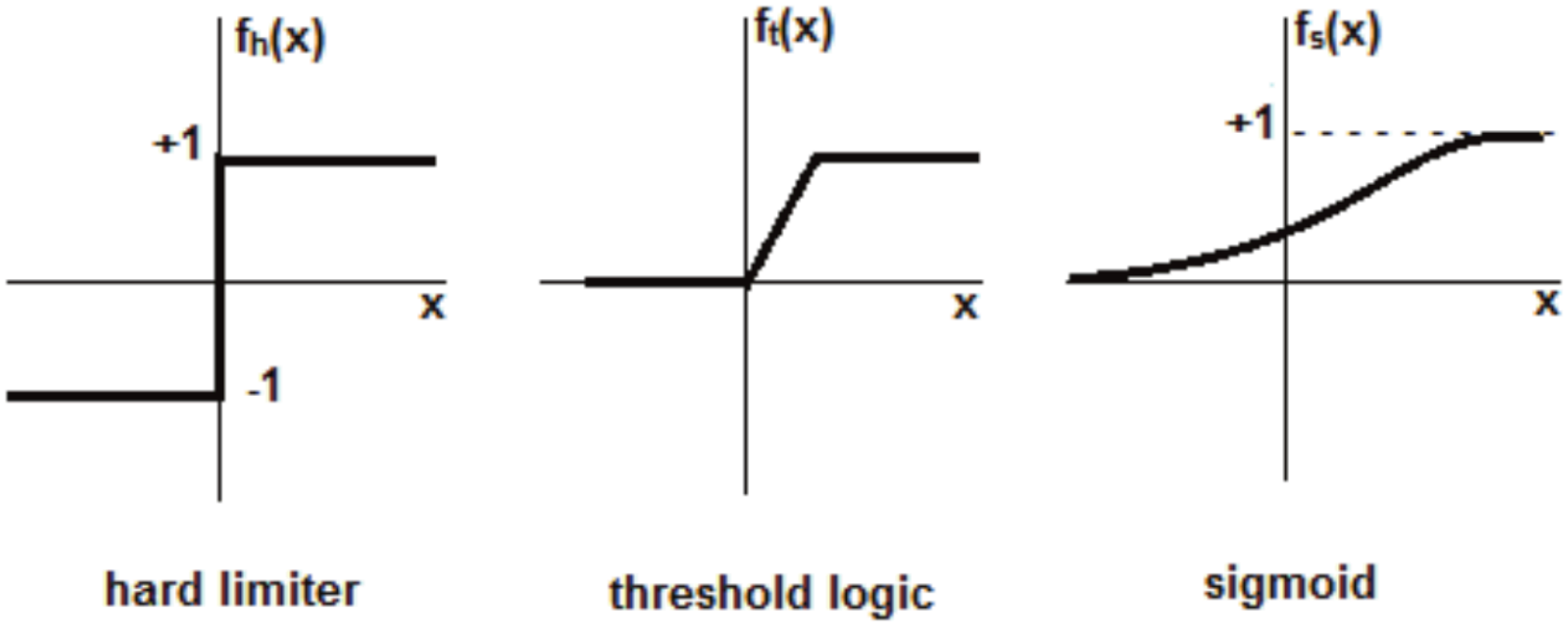
Activation functions

- Applied to node inputs to produce node output



Activation Functions

An artificial neural network neuron Each neuron has an activation function. the input for a neuron activation function is the weighted sum of the output of the other neurons and the output of the activation function is calculated by some functions such as hard limit, threshold logic, sigmoid



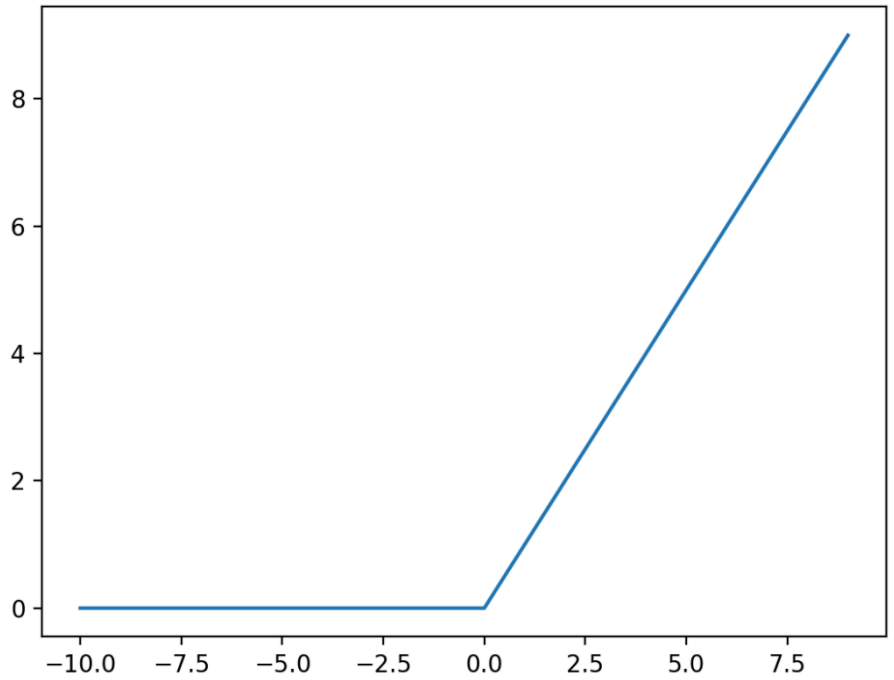
Nonlinear Activation Functions

When building a model and training a neural network, the selection of activation functions is critical. Experimenting with different activation functions for different problems will allow you to achieve much better results.

- Sigmoid /Logistic
- TanH
- ReLU (Rectified Linear Unit)
- Leaky ReLU
- Parametric ReLU
- Softmax
- Swish

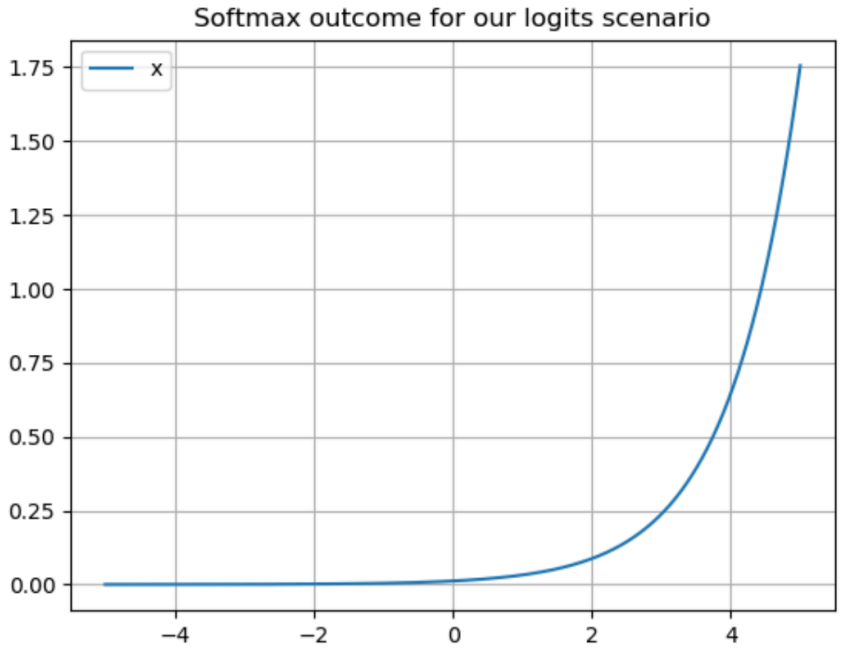
https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/

ReLU



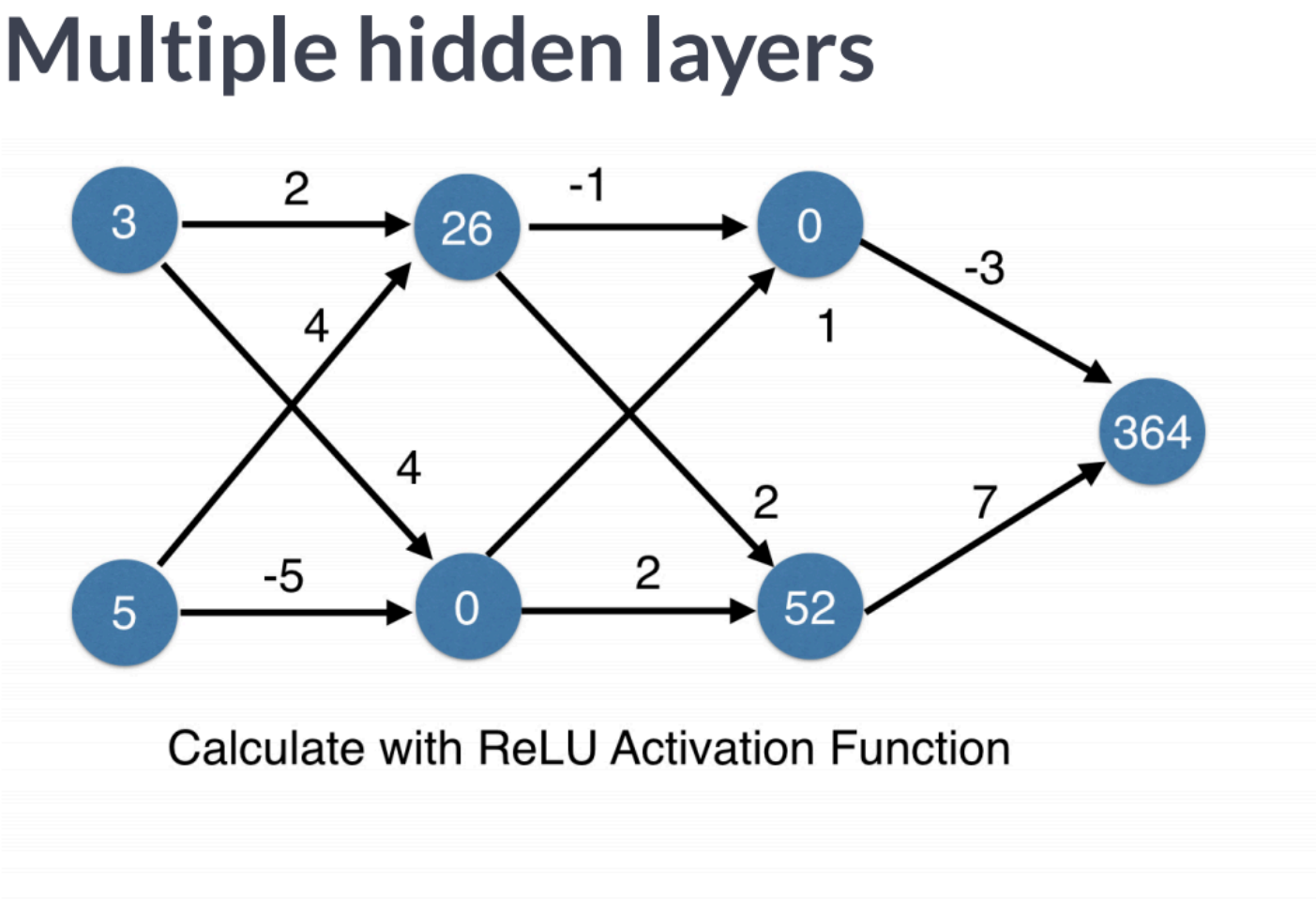
All negative values and zero inputs are snapped to 0.0, whereas the positive outputs are returned as-is, resulting in a linearly increasing slope, given that we created a linearly increasing series of positive values

Softmax

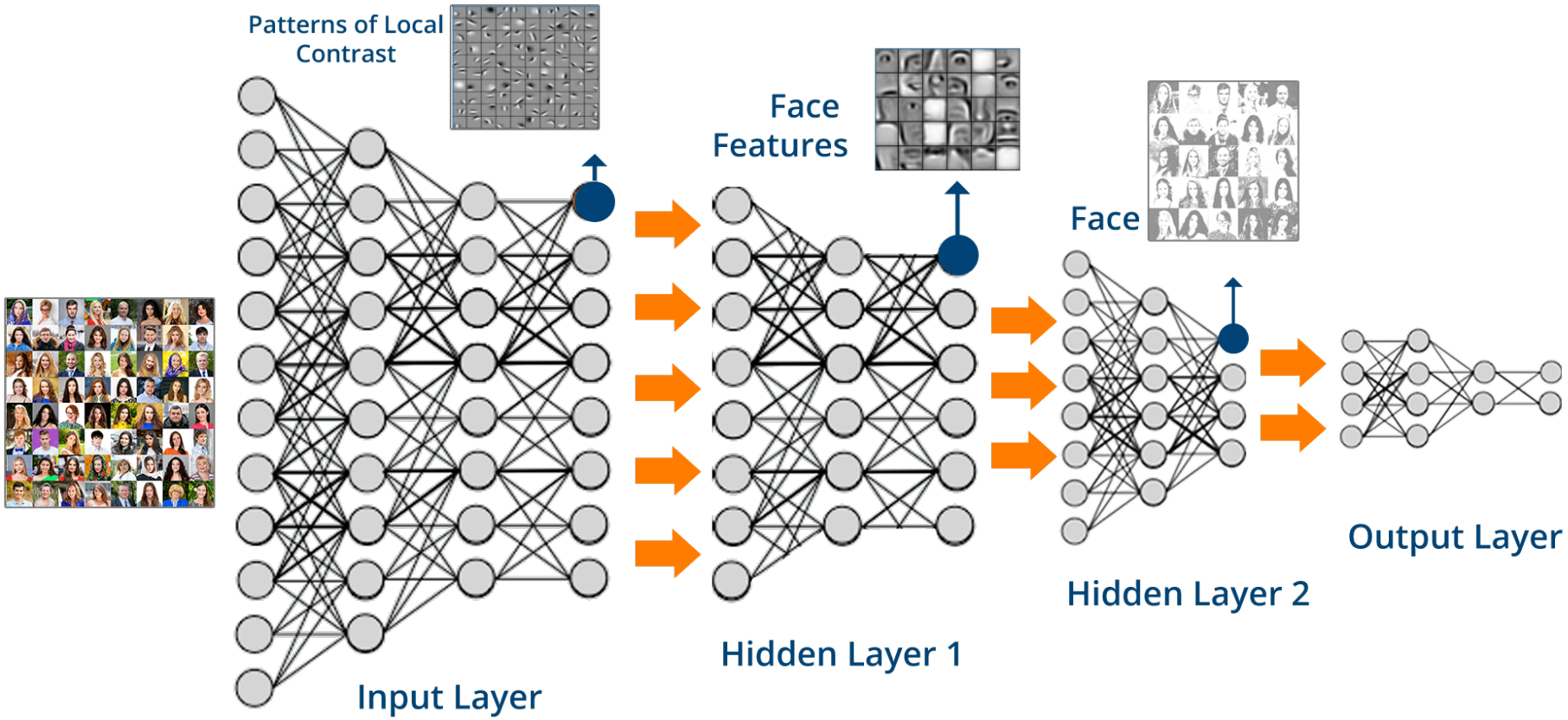


- The softmax function is a generalization of the logistic function to multiple dimensions. It is used in multinomial logistic regression and is often used as the last activation function of a neural network to normalize the output of a network to a probability distribution over predicted output classes.
- Softmax is used as the activation function for multi-class classification problems where class membership is required on more than two class labels.

Deeper Networks



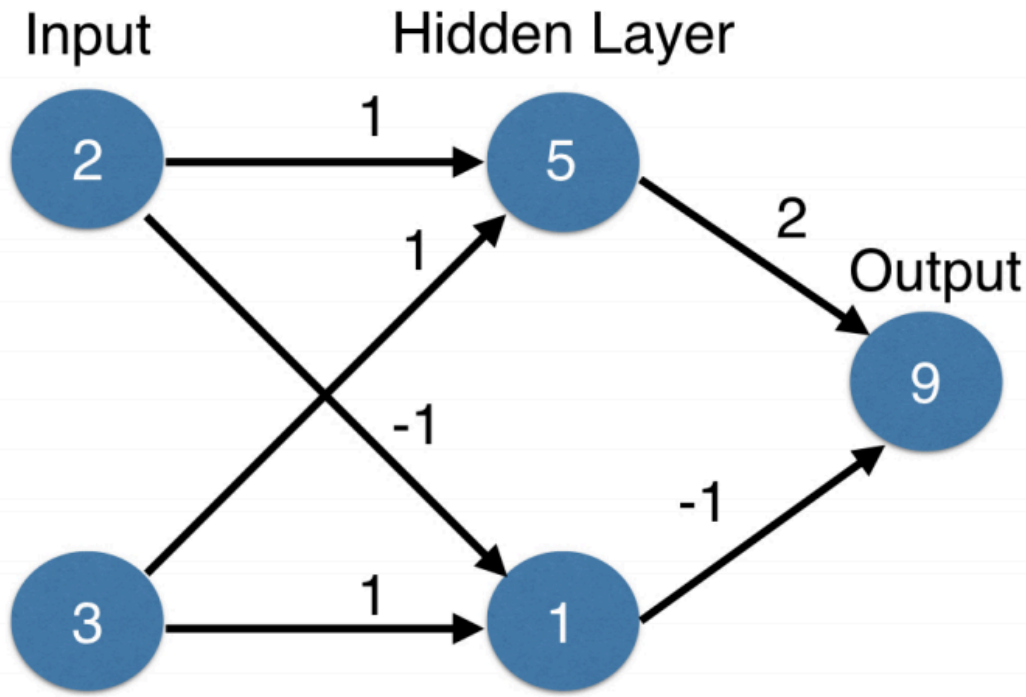
Deep Neural Networks for Computer Vision



Optimization

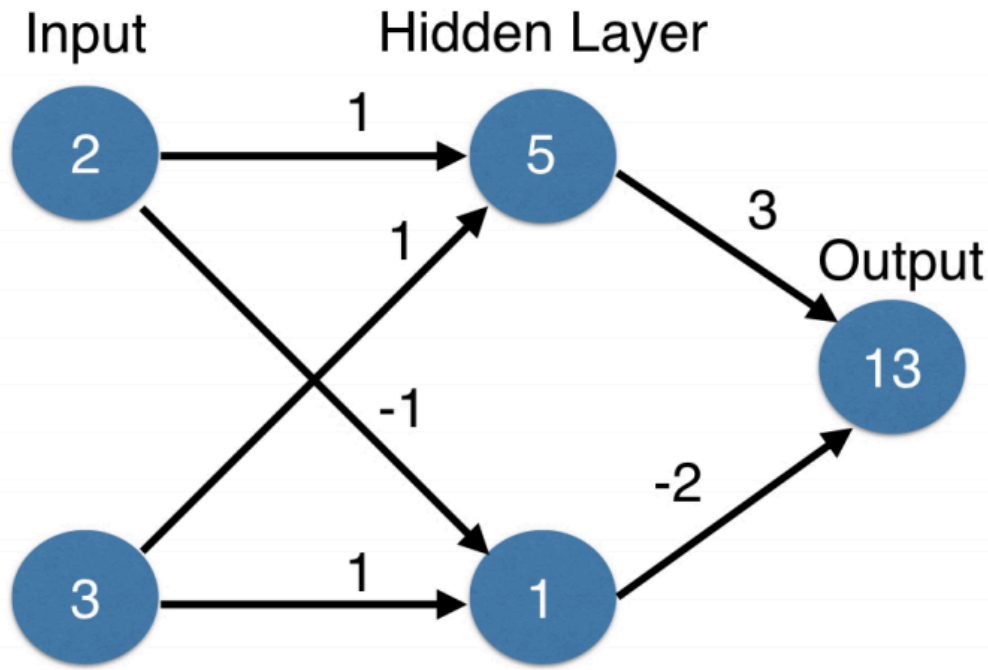
- When talking about optimization in the context of neural networks, we are discussing **non-convex optimization**.
- *Convex optimization* involves a function in which there is only one optimum, corresponding to the global optimum (maximum or minimum).
- For a neural network, the curve or surface that we are talking about is the loss surface. Since we are trying to minimize the prediction error of the network, we are interested in finding the global minimum on this loss surface — this is the aim of neural network training.

A baseline neural network



- Actual Value of Target: 13
- Error: Predicted - Actual = -4

A baseline neural network



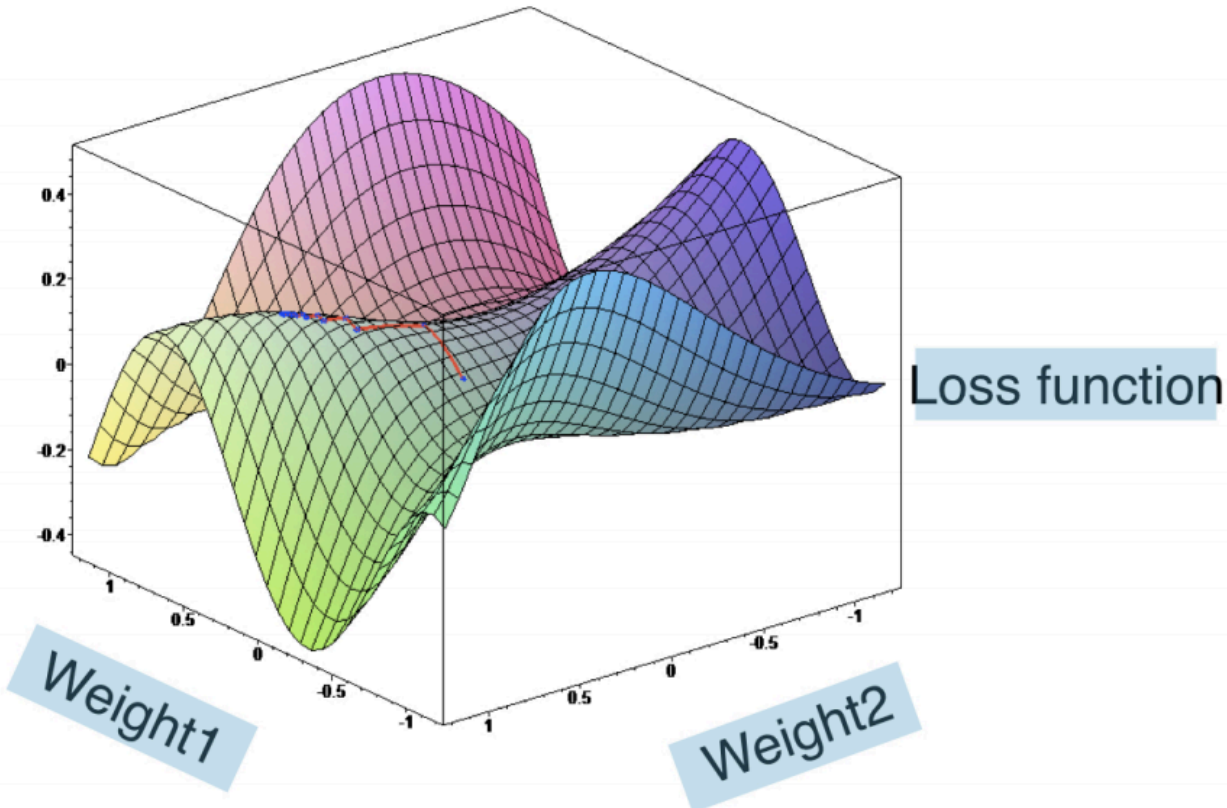
- Actual Value of Target: 13
- Error: Predicted - Actual = 0

Squared error loss function

Prediction	Actual	Error	Squared Error
10	20	-10	100
8	3	5	25
6	1	5	25

- Total Squared Error: 150
- Mean Squared Error: 50

Loss function

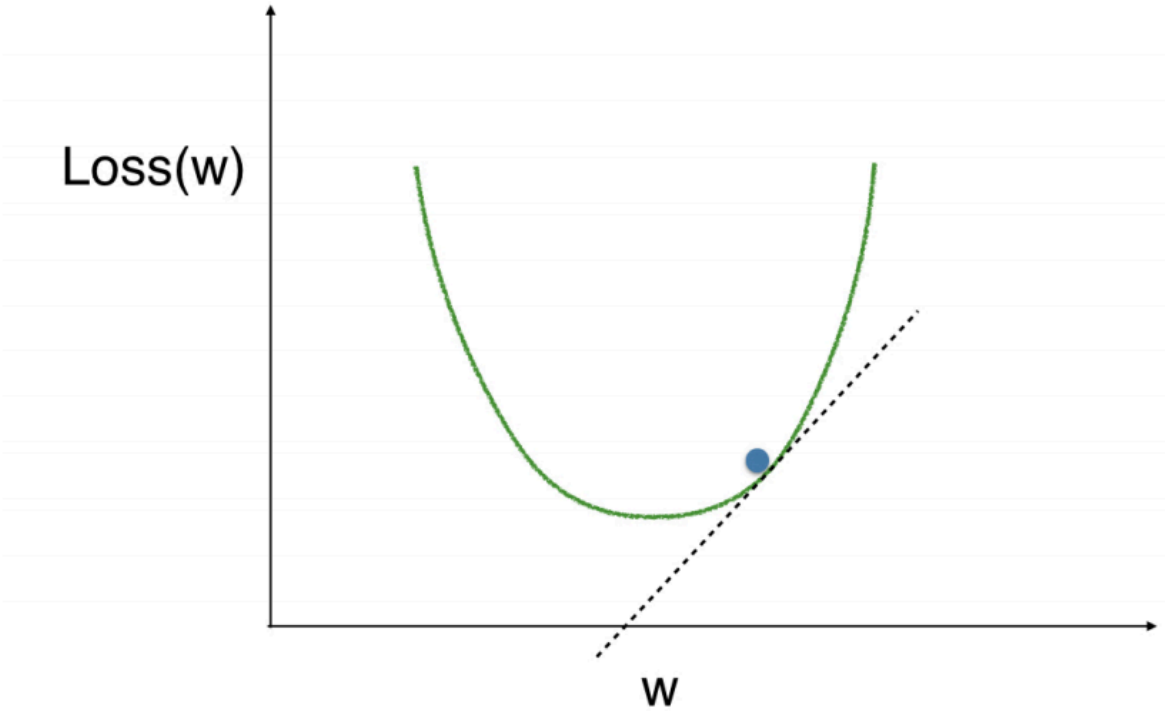


Loss Functions for Neural Networks

- Regression Loss Functions
  - Mean Squared Error Loss
  - Mean Squared Logarithmic Error Loss
  - Mean Absolute Error Loss
- Binary Classification Loss Functions
  - Binary Cross-Entropy
  - Hinge Loss
  - Squared Hinge Loss
- Multi-Class Classification Loss Functions
  - Multi-Class Cross-Entropy Loss
  - Sparse Multiclass Cross-Entropy Loss
  - Kullback Leibler Divergence Loss

https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/

Gradient descent



- If the slope is positive:
  - Going opposite the slope means moving to lower numbers
  - Subtract the slope from the current value
  - Too big a step might lead us astray
- Solution: learning rate
  - Update each weight by subtracting learning rate * slope

Slope calculation example

- To calculate the slope for a weight, need to multiply:
  - Slope of the loss function w.r.t value at the node we feed into
  - The value of the node that feeds into our weight
  - Slope of the activation function w.r.t value we feed into

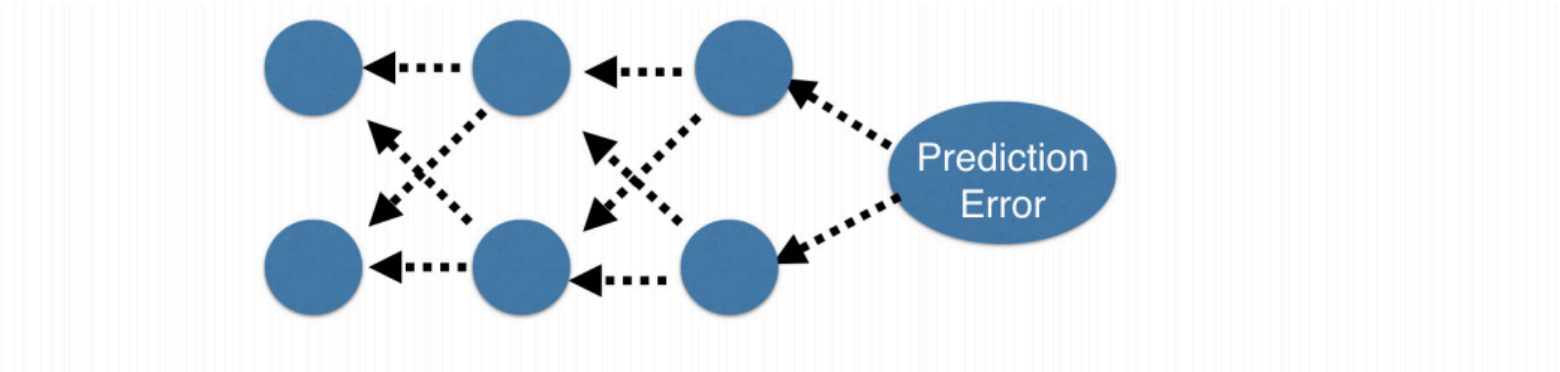
- `2 * -4 * 3`
- `-24`
- If learning rate is `0.01`, the new weight would be
- `2 - 0.01(-24) = 2.24`

Optimization Functions

Commonly used optimization functions for Neural Network

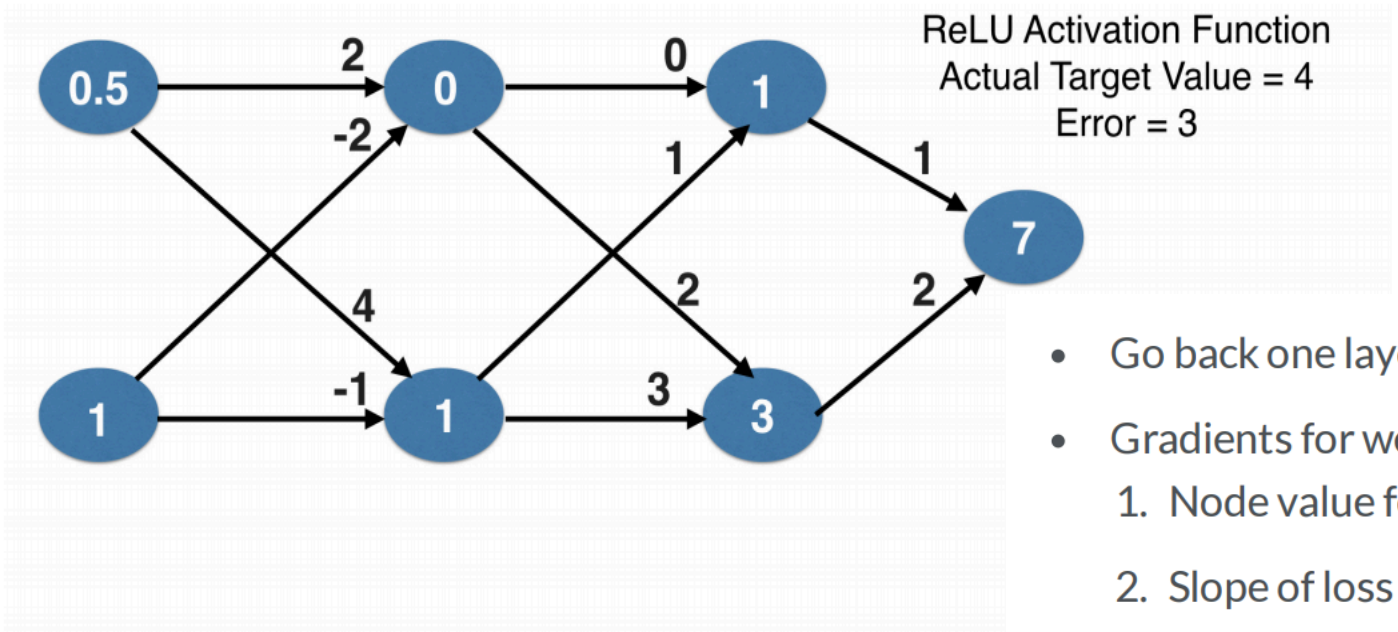
- Stochastic Gradient Descent (SGD)
  - Calculating slopes on only a subset of the data (a batch)
  - Start over from the beginning once all data is used
  - Each time through the training data is called an epoch
- AdaGrad
- RMSProp
- Adam

Backpropagation



- Allows gradient descent to update all weights in neural network (by getting gradients for all weights)
- Comes from chain rule of calculus
- Important to understand the process, but you will generally use a library that implements this

Backpropagation process



- Go back one layer at a time
- Gradients for weight is product of:
  1. Node value feeding into that weight
  2. Slope of loss function w.r.t node it feeds into
  3. Slope of activation function at the node it feeds into
- Need to also keep track of the slopes of the loss function w.r.t node values
- Slope of node values are the sum of the slopes for all weights that come out of them

Keras Model

Model building steps

- Specify Architecture
- Compile
- Fit
- Predict

Model specification

```
import numpy as np
from keras.layers import Dense
from keras.models import Sequential

predictors = np.loadtxt('predictors_data.csv', delimiter=',')
n_cols = predictors.shape[1]

model = Sequential()
model.add(Dense(100, activation='relu', input_shape = (n_cols,)))
model.add(Dense(100, activation='relu'))
model.add(Dense(1))
```

Why you need to compile your model

- Specify the optimizer
  - Many options and mathematically complex
  - "Adam" is usually a good choice
- Loss function
  - "mean_squared_error" common for regression

Compiling a model

```
n_cols = predictors.shape[1]
model = Sequential()
model.add(Dense(100, activation='relu', input_shape=(n_cols,)))
model.add(Dense(100, activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mean_squared_error')
```

What is fitting a model

- Applying backpropagation and gradient descent with your data to update the weights
- Scaling data before fitting can ease optimization

Fitting a model

```
n_cols = predictors.shape[1]
model = Sequential()
model.add(Dense(100, activation='relu', input_shape=(n_cols,)))
model.add(Dense(100, activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(predictors, target)
```

Saving, reloading and using your Model

```
from keras.models import load_model
model.save('model_file.h5')
my_model = load_model('my_model.h5')
predictions = my_model.predict(data_to_predict_with)
probability_true = predictions[:,1]
```

Verifying model structure

```
my_model.summary()
```

```
_________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=========================================================================================
dense_1 (Dense)              (None, 100)               1100        dense_input_1[0][0]
_________________________________________________________________________________________
dense_2 (Dense)              (None, 100)               10100       dense_1[0][0]
_________________________________________________________________________________________
dense_3 (Dense)              (None, 100)               10100       dense_2[0][0]
_________________________________________________________________________________________
dense_4 (Dense)              (None, 2)                 202         dense_3[0][0]
=========================================================================================
Total params: 21,502
Trainable params: 21,502
Non-trainable params: 0
```

Validation in deep learning

- Commonly use validation split rather than cross-validation
- Deep learning widely used on large datasets
- Single validation score is based on large amount of data, and is reliable
- Repeated training from cross-validation would take long time

Model validation

```
model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics=['accuracy'])
model.fit(predictors, target, validation_split=0.3)
```

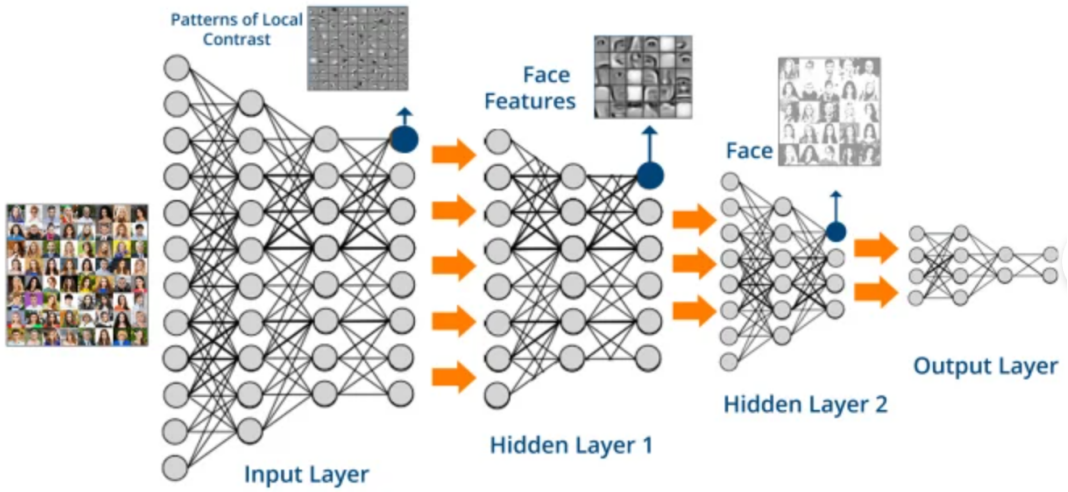
```
Epoch 1/10
89648/89648 [=====] - 3s - loss: 0.7552 - acc: 0.5775 - val_loss: 0.6969 - val_acc: 0.5561
Epoch 2/10
89648/89648 [=====] - 4s - loss: 0.6670 - acc: 0.6004 - val_loss: 0.6580 - val_acc: 0.6102
...
Epoch 8/10
89648/89648 [=====] - 5s - loss: 0.6578 - acc: 0.6125 - val_loss: 0.6594 - val_acc: 0.6037
Epoch 9/10
89648/89648 [=====] - 5s - loss: 0.6564 - acc: 0.6147 - val_loss: 0.6568 - val_acc: 0.6110
Epoch 10/10
89648/89648 [=====] - 5s - loss: 0.6555 - acc: 0.6158 - val_loss: 0.6557 - val_acc: 0.6126
```

Experimentation

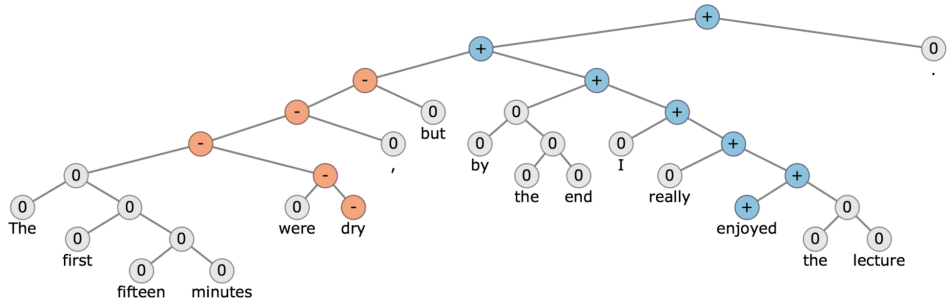
- Experiment with different architectures
- More layers
- Fewer layers
- Layers with more nodes
- Layers with fewer nodes
- Creating a great model requires experimentation

Applications for Artificial Neural Networks

Computer Vision



Natural Language Processing (NLP)



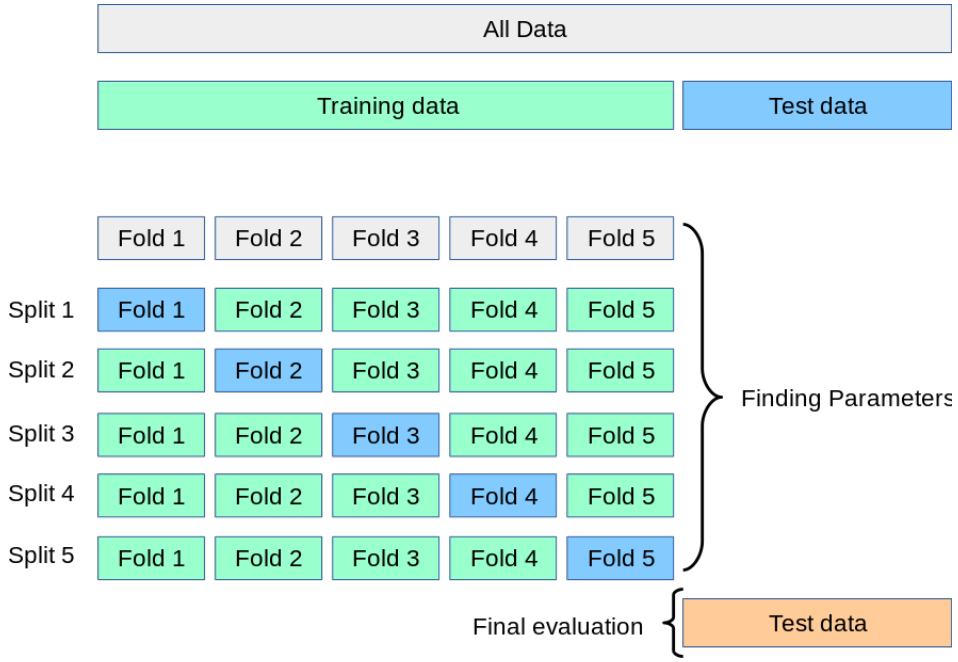
Model Evaluation and Tuning

Streamlining workflows with pipelines

- Pipeline is a handy tool to allow us to fit a model including several transformation steps and apply it to make predictions about new data.
- Pipeline simplifies the Python code, for example, we can chain the transformation and fitting steps to avoid going through training and test dataset separately.

K-fold cross-validation

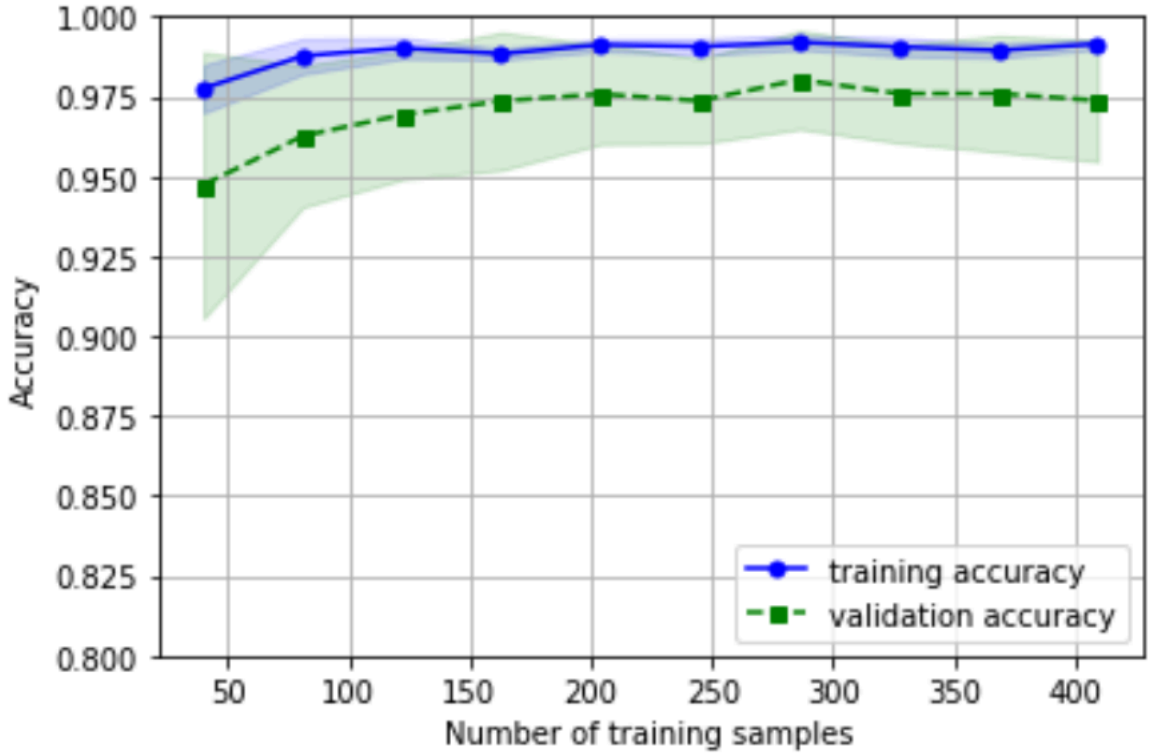
- K-fold cross-validation helps us to obtain reliable estimates of the model's generalization error, that is, how well the model performs on unseen data.
- In k-fold cross-validation, we randomly split the training dataset into k folds without replacement, where k-1 folds are used for training and the one fold is used for testing.
- This procedure repeated k times, so that we obtain k models, the performances will be averaged.



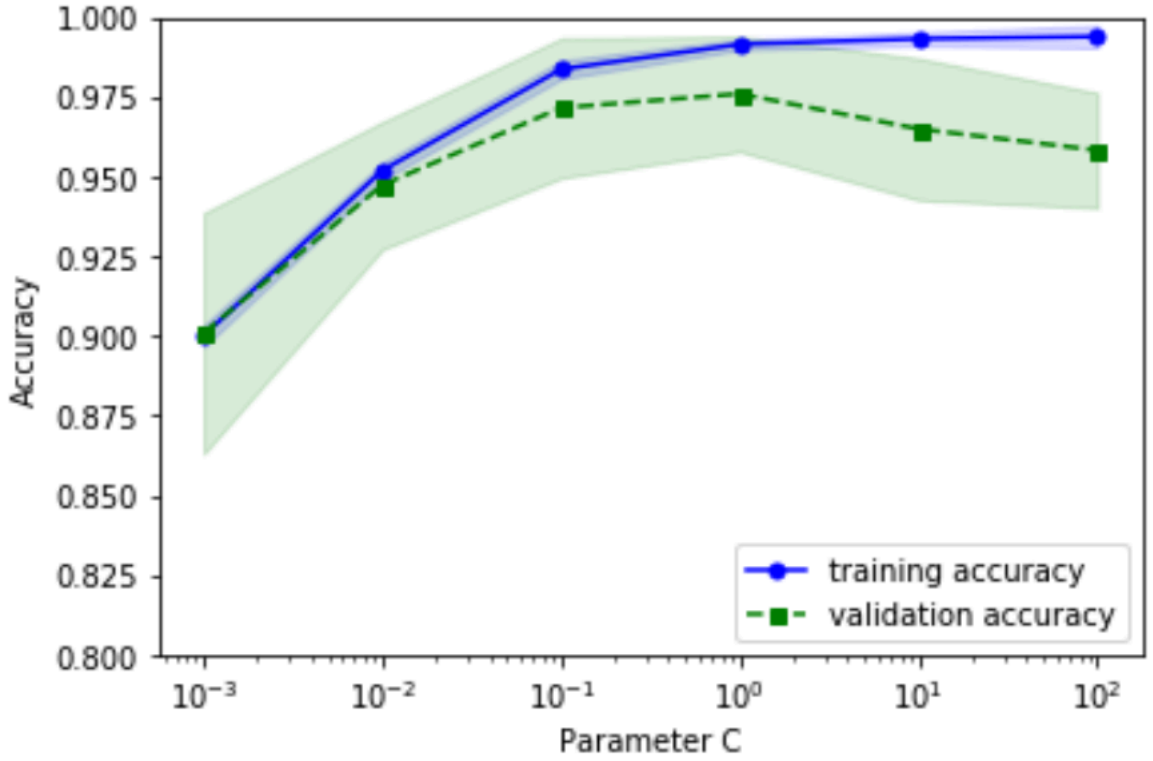
K-fold cross-validation

- The advantage of k-fold cross-validation is that each sample point will be in the training and testing dataset once. This yields a lower variance.
- The standard value for k is 10, when training set is small, we can increase the value of k so more data will be used for training in each iteration, which results in a lower bias. When datasets are large, we can decrease the value of k to reduce the computational cost.

Model evaluations with training and testing curves



Inverse Regularization Parameter C



Fine-tuning machine learning models via grid search

- In machine learning, there are 2 types of parameters:
  - The ones learned from training, e.g. weights from logistic regression
  - The parameters of a learning algorithm that are optimized (hyperparameters for tuning), for example, the inverse regularization parameter C or the depth of a decision tree.
- Grid Search is a powerful hyperparameter optimization technique.
  - It is a brute-force exhaustive search paradigm for the best hyperparameter combination from a list of specified values.

Fine-tuning deep learning model via Hyperas

A very simple convenience wrapper around hyperopt for fast prototyping with keras models.

Hyperas lets you use the power of hyperopt without having to learn the syntax of it. Instead, just define your keras model as you are used to but use a simple template notation to define hyper-parameter ranges to tune.

https://github.com/maxpumperla/hyperas