Deep Neural Networks

What exactly is deep leanring?

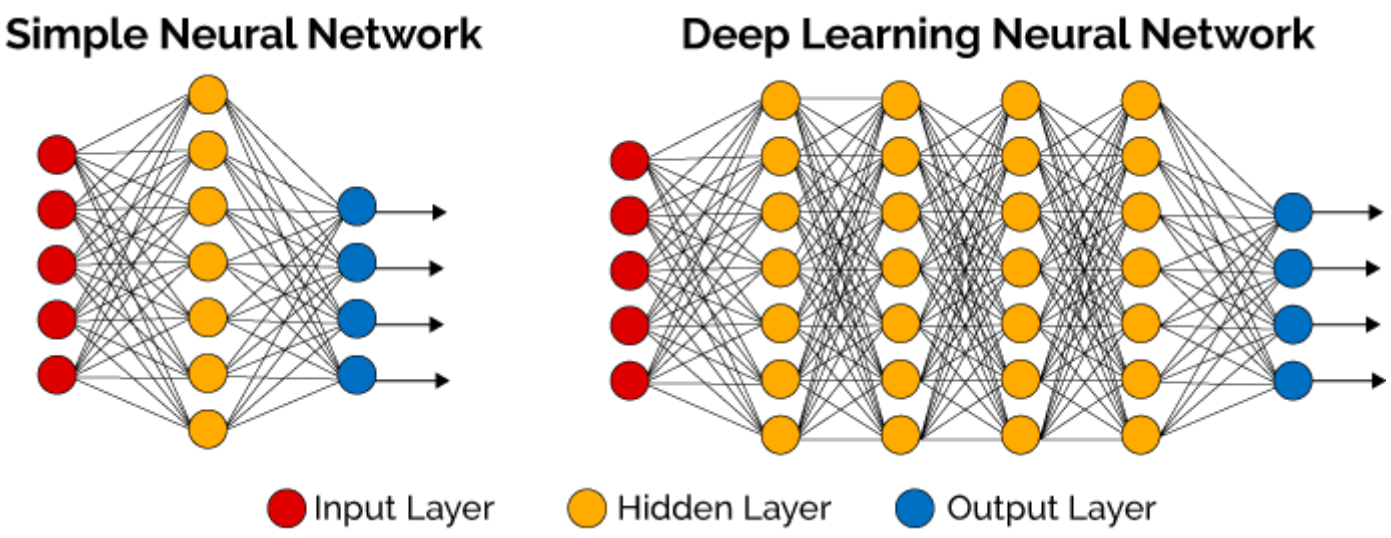
'Deep Learning' means using a neural network with several layers of nodes between input and output

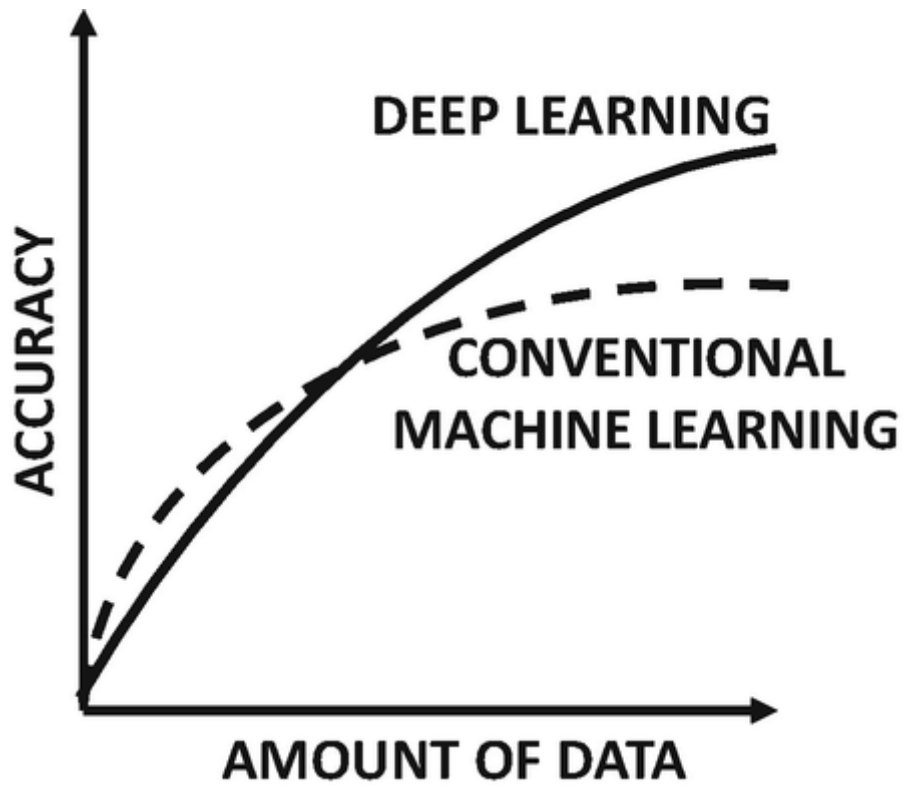
Why is it generally better than other methods on image, speech and certain other types of data?

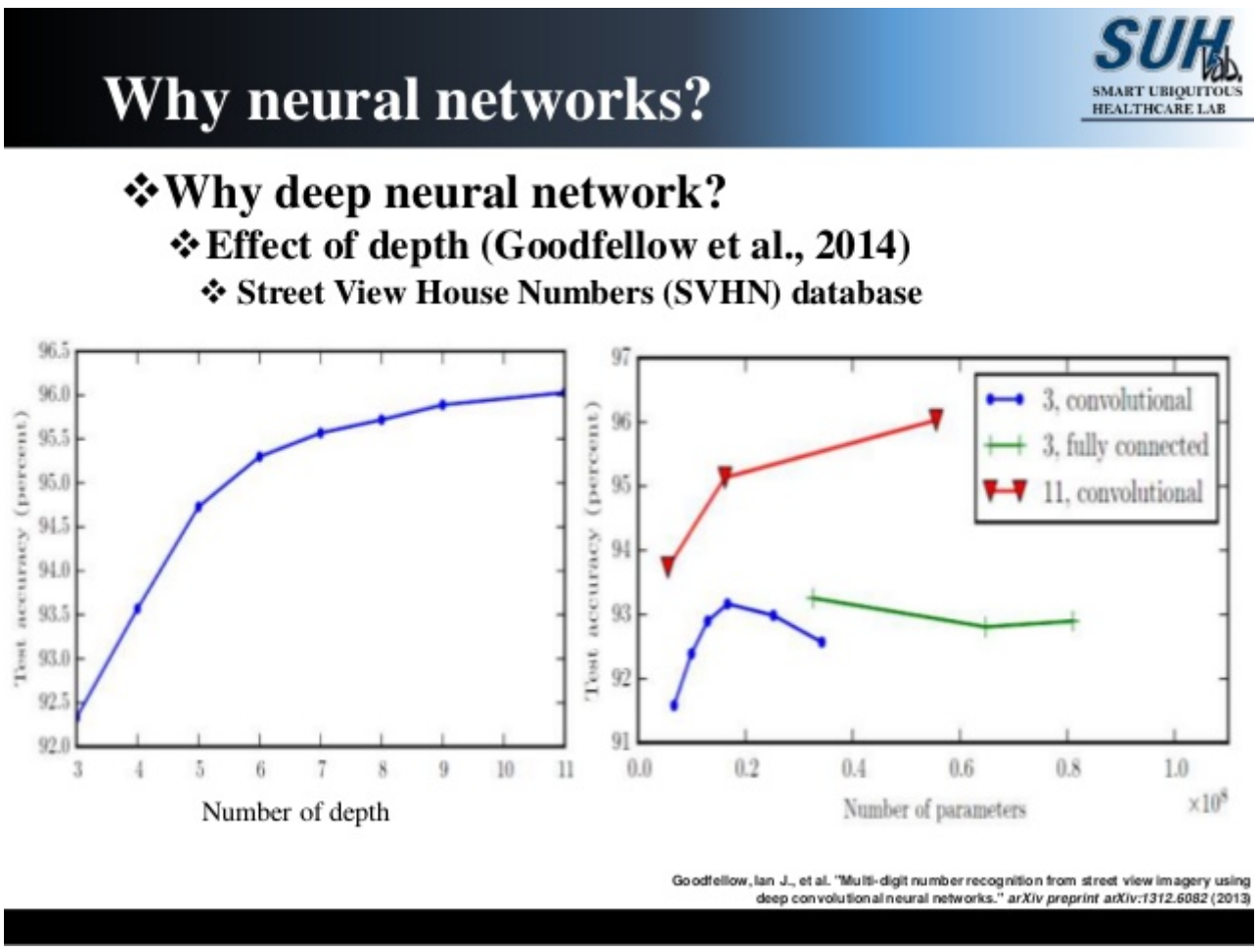
The series of layers between input & output do feature identification and processing in a series of stages, just as our brains seem to.

Multilayer neural networks have been around for 25 years. What's actually new?

We have always had good algorithms for learning the weights in networks with 1 hidden layer but these algorithms are not good at learning the weights for networks with more hidden layers what's new is: algorithms for training many-layer networks







In practice deeper networks usually represent mode complex functions with less total neurons (and therefore less parameters)

- Compactly express nice, smooth functions that fit well with the statistical properties of data we encounter in practice
- Easy to learn using the optimization algorithms we have available


```
import numpy as np
import random as rn
rn.seed(11)
np.random.seed(11)
import csv
import pandas as pd
from sklearn.datasets import make_blobs,make_moons, load_iris, make_circles, load_boston
import matplotlib.pyplot as plt
from numpy import loadtxt
from sklearn.model_selection import train_test_split

# first neural network with keras tutorial
from numpy import loadtxt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Dropout, Conv1D, BatchNormalization, Flatten, Conv2D, Input
from tensorflow.keras import backend as K


def plot_history(history):
  plt.plot(history.history['loss'])
  plt.plot(history.history['val_loss'])
  plt.title('model loss')
  plt.ylabel('loss')
  plt.xlabel('epoch')
  plt.legend(['train', 'test'], loc='upper left')
  plt.show()
  return
```

Regression problem

A regression problem is when the output variable is a real or continuous value.

Prediction of heating and cooling capacity is important for the planning and management of energy systems.

Energy analysis using 12 different building shape

Given:

- X1 Relative Compactness
- X2 Surface Area
- X3 Wall Area
- X4 Roof Area
- X5 Overall Height
- X6 Orientation
- X7 Glazing Area
- X8 Glazing Area Distribution

Predict:

- y1 Heating Load
- y2 Cooling Load

V. V. Mokeev, "Prediction of Heating Load and Cooling Load of Buildings Using Neural Network," 2019 International Ural Conference on Electrical Power Engineering (UralCon), Chelyabinsk, Russia, 2019, pp. 417-421, doi: 10.1109/URALCON.2019.8877655.

Neural Network:

- Number of neurons
- Number of layers
- Types of layers

Mean Square Error Loss

$$MSE = \frac{1}{n}\sum\left(y - \hat{y}\right)^2$$

The square of the difference between actual and predicted

Adaptive Moment Estimation (Adam)

Adam is an adaptive learning rate optimization algorithm that's been designed specifically for training deep neural networks.

- Adam is different to classical stochastic gradient descent.
- The method computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients.
- Helps in convergence
- In practice Adam is currently recommended as the default algorithm to use. However, it is often also worth trying SGD+Nesterov Momentum as an alternative.

https://arxiv.org/pdf/1412.6980.pdf

$$For\ each\ Parameter\ w^j$$

$$(j\ subscript\ dropped\ for\ clarity)$$

$$\nu_t = \beta_1 * \nu_{t-1} - (1 - \beta_1) * g_t$$

$$s_t = \beta_2 * s_{t-1} - (1 - \beta_2) * g_t^2$$

$$\Delta\omega_t = -\eta \frac{\nu_t}{\sqrt{s_t + \epsilon}} * g_t$$

$$\omega_{t+1} = \omega_t + \Delta\omega_t$$

$\eta : Initial\ Learning\ rate$

$g_t : Gradient\ at\ time\ t\ along\ \omega^j$

$\nu_t : Exponential\ Average\ of\ gradients\ along\ \omega_j$

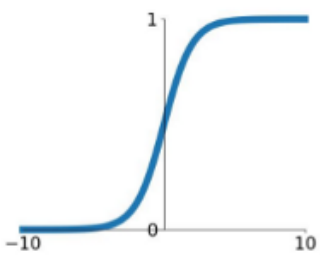
$s_t : Exponential\ Average\ of\ squares\ of\ gradients\ along\ \omega_j$

$\beta_1, \beta_2 : Hyperparameters$

Activation Functions

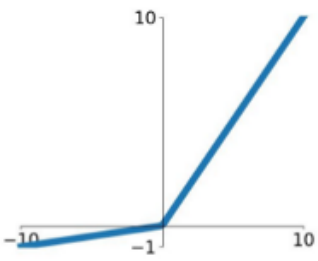
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



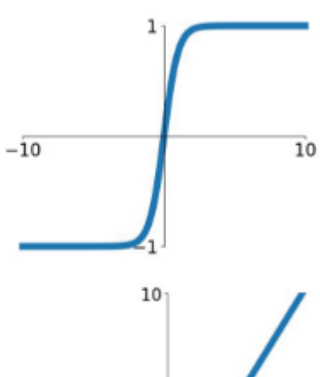
Leaky ReLU

$\max(0.1x, x)$



tanh

$\tanh(x)$

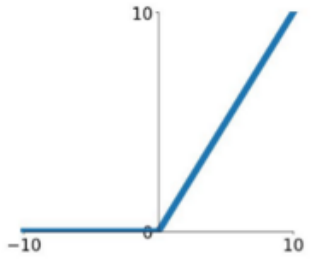


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

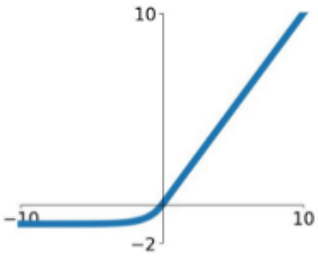
ReLU

$\max(0, x)$



ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



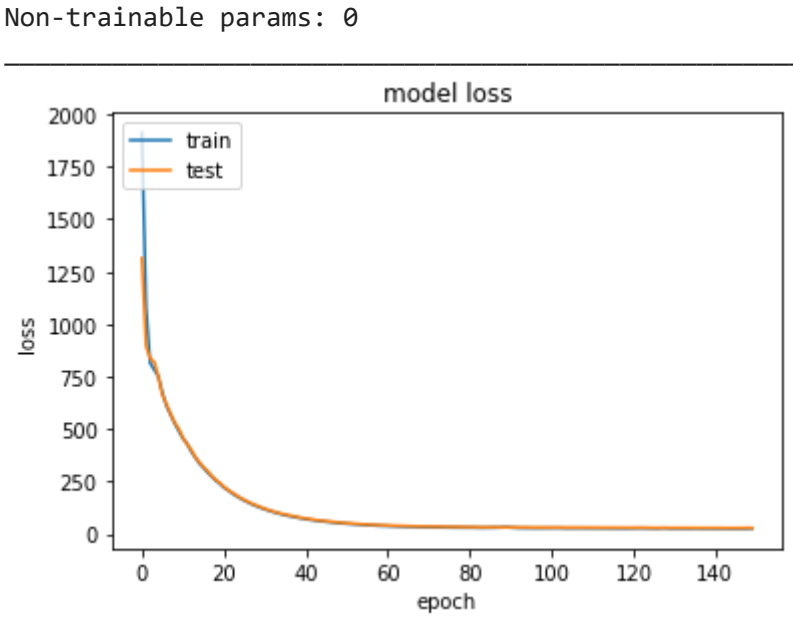

```
#Loading the data
datainput = pd.read_csv('ENB2012_data.csv')
# Print the top 5 records
print(datainput[0:5],"\n")

# Print the complete shape of the dataset
print("Shape of Complete Data Set")
print(datainput.shape,"\n")
#separating features(X) and label(y)
X = datainput.iloc[:, :-2].values

# Select the last column of all rows
```

```
Y = datainput.iloc[:,-2:].values

#train_test_split method
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2)
print(X_train.shape)
print(Y_train.shape)
print(X_test.shape)
print(Y_test.shape)

# Neural Network Parameters
E=150
BS=100
```

```
      X1     X2     X3      X4   X5  X6   X7  X8     Y1     Y2
0   0.98  514.5  294.0  110.25  7.0   2  0.0   0  15.55  21.33
1   0.98  514.5  294.0  110.25  7.0   3  0.0   0  15.55  21.33
2   0.98  514.5  294.0  110.25  7.0   4  0.0   0  15.55  21.33
3   0.98  514.5  294.0  110.25  7.0   5  0.0   0  15.55  21.33
4   0.90  563.5  318.5  122.50  7.0   2  0.0   0  20.84  28.28

Shape of Complete Data Set
(768, 10)

(614, 8)
(614, 2)
(154, 8)
(154, 2)
```

```
# small layer
K.clear_session()
model = Sequential()

model.add(Dense(24, input_dim=8, activation='relu'))
model.add(Dense(2, activation='linear'))
model.summary()

# compile the keras model
model.compile(loss='mse', optimizer='adam')

# fit the keras model on the dataset
history = model.fit(X_train, Y_train, epochs=E, batch_size=BS, validation_data=(X_test,Y_test),verbose=0)
plot_history(history)
print('Minimum Validation Loss: ',min(history.history['val_loss']))
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 24)                216
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 50
=================================================================
Total params: 266
Trainable params: 266
Non-trainable params: 0
_________________________________________________________________
```

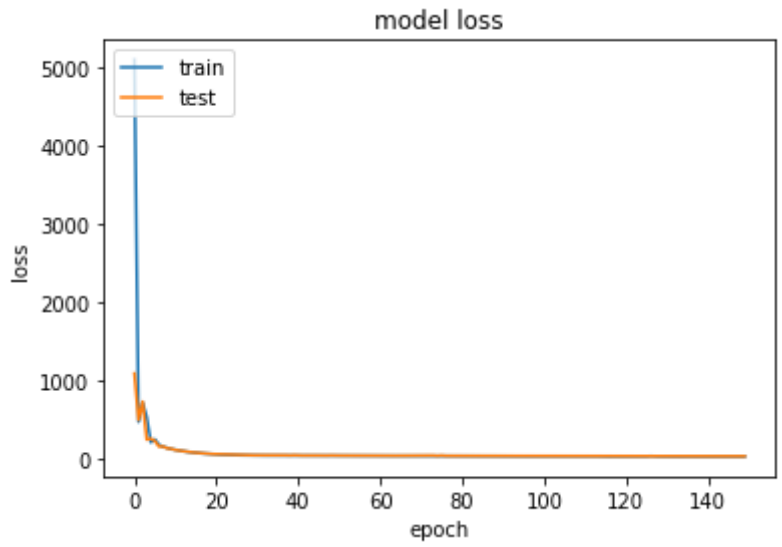



```
Minimum Validation Loss:  28.75547981262207
```

```
# 100 neuron layer
K.clear_session()
model = Sequential()
model.add(Dense(24+48+64, input_dim=8, activation='relu'))
model.add(Dense(2, activation='linear'))
model.summary()

# compile the keras model
model.compile(loss='mse', optimizer='adam')
```

```
model.compile(loss='mse', optimizer='adam')

# fit the keras model on the dataset
# fit the keras model on the dataset
history = model.fit(X_train, Y_train, epochs=E, batch_size=BS,validation_data=(X_test,Y_test),verbose=0)
plot_history(history)
print('Minimum Validation Loss: ',min(history.history['val_loss']))
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 136)               1224
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 274
=================================================================
Total params: 1,498
Trainable params: 1,498
Non-trainable params: 0
_________________________________________________________________
```




```
Minimum Validation Loss:  20.886066436767578
```

```
# deeper model with 3 layers
K.clear_session()
model = Sequential()
model.add(Dense(24, input_dim=8, activation='relu'))
model.add(Dense(48, activation='relu'))
model.add(Dense(64,activation='relu'))
model.add(Dense(2, activation='linear'))
model.summary()

# compile the keras model
model.compile(loss='mse', optimizer='adam')

# fit the keras model on the dataset
history = model.fit(X_train, Y_train, epochs=E, batch_size=BS,validation_data=(X_test,Y_test),verbose=0)
plot_history(history)
print('Minimum Validation Loss: ',min(history.history['val_loss']))
```

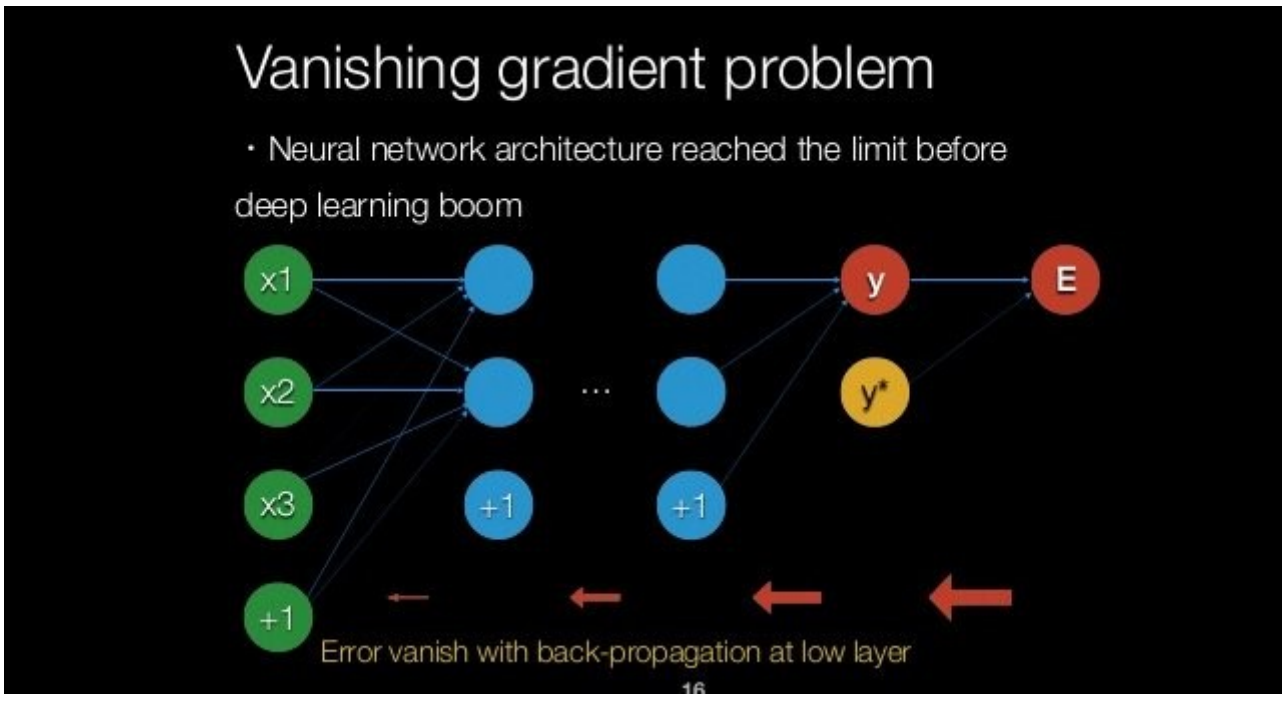
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
```

```
# 100 layer model
K.clear_session()
model = Sequential()
model.add(Dense(24, input_dim=8, activation='relu'))
for i in range(100):
  model.add(Dense(48,activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(2, activation='linear'))
model.summary()

# compile the keras model
model.compile(loss='mse', optimizer='adam')

# fit the keras model on the dataset
history = model.fit(X_train, Y_train, epochs=E, batch_size=BS,validation_data=(X_test,Y_test),verbose=0)
plot_history(history)
print('Minimum Validation Loss: ',min(history.history['val_loss']))
```


Why not simply add more layers?

- We cannot make networks arbitrary complex?

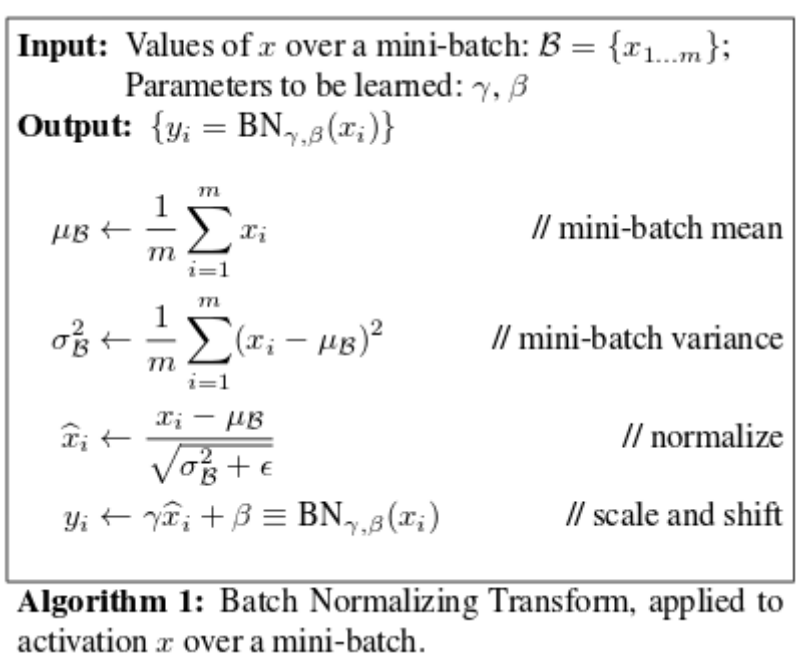
- Why not just go deeper and get better?
    - No Structure!!
    - It is just brute force!
    - Optimization becomde harder
    - Performance plateaus/drops



How to improve performance of deeper networks

Batch Normalization

- It was motivated by the problem of internal covariance shift, where changes in the distribution of the inputs of each layer affects the l
- It tries to make the output of a layer have a mean of 0 and unit variance by normalizing the output based on statistics of the batch earning rate of the network.



Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Advantages of BN:

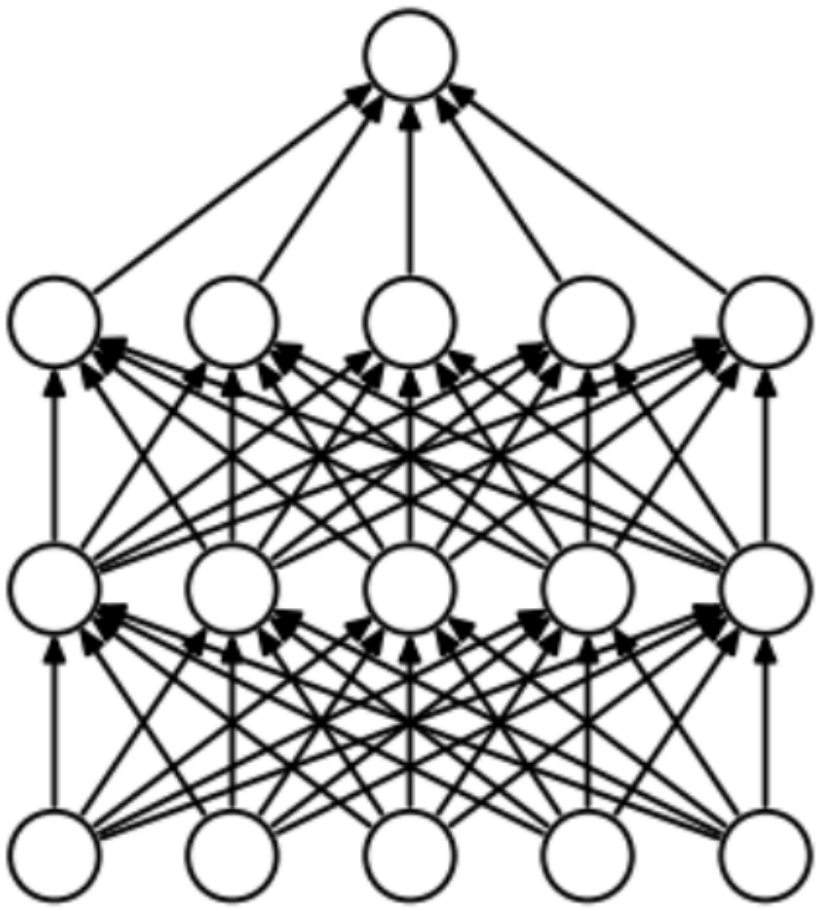
- Very deep nets are much easier to train, more stable gradients
- A much larger range of hyperparameters works similarly when using BN

A batch normalization layer normalizes each input channel across a mini-batch. To speed up training of convolutional neural networks and reduce the sensitivity to network initialization, use batch normalization layers between convolutional layers and nonlinearities, such as ReLU layers.

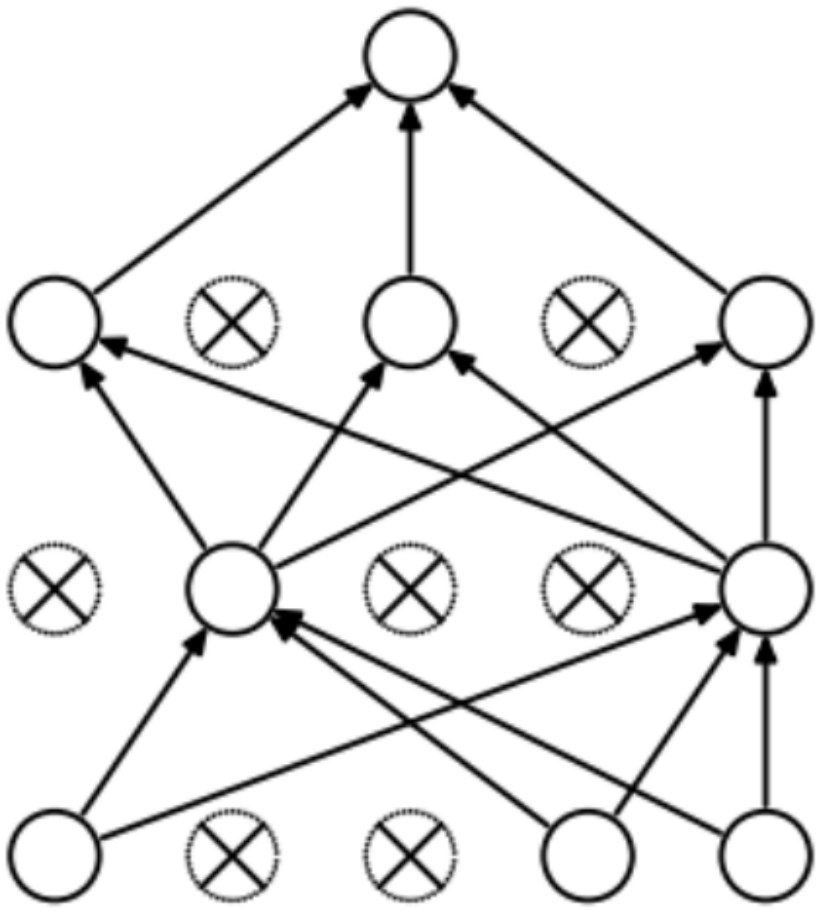
Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift [Ioffe and Szegedy 2015]

Dropout

- Disable a random set of neurons (typically 20%-50%)
    - use half the network = half capacity
    - Create Redundant representations
    - Base your scores on more features
    - Consider it as a model ensemble
- Dropout reduces the effective capacity of a model -larger models, more training time



(a) Standard Neural Net



(b) After applying dropout.

```
# deeper model with 3 layers
X_train = np.squeeze(X_train)
X_test = np.squeeze(X_test)

K.clear_session()
model = Sequential()
model.add(Dense(24, input_dim=8, activation='relu'))
model.add(BatchNormalization())
model.add(Dense(48, activation='relu'))
model.add(Dropout(0.25))
model.add(BatchNormalization())
model.add(Dense(64,activation='relu'))
model.add(BatchNormalization())
model.add(Dense(2, activation='linear'))
model.summary()

# compile the keras model
model.compile(loss='mse', optimizer='adam')

# fit the keras model on the dataset
history = model.fit(X_train, Y_train, epochs=E, batch_size=BS,validation_data=(X_test,Y_test),verbose=0)
plot_history(history)
print('Minimum Validation Loss: ',min(history.history['val_loss']))
```

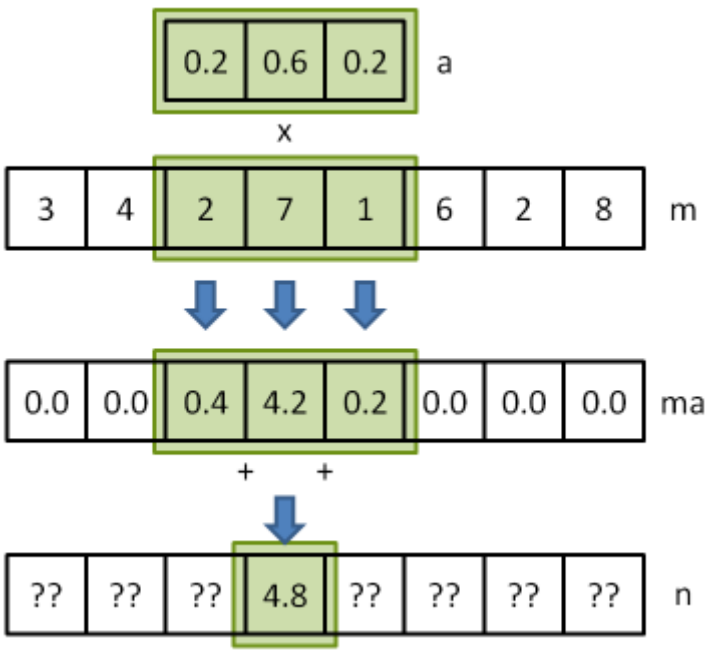
Convolutional Layers

Convolutional Neural Network (CNN) models were developed for image classification, in which the model accepts a two-dimensional input representing an image's pixels and color channels, in a process called feature learning.

This same process can be applied to one-dimensional sequences of data. The model extracts features from sequences data and maps the internal features of the sequence. A 1D CNN is very effective for deriving features from a fixed-length segment of the overall dataset, where it is not so important where the feature is located in the segment.

1D Convolutional Neural Networks work well for:

- Analysis of a time series of sensor data.
- Analysis of signal data over a fixed-length period, for example, an audio recording.
- Natural Language Processing (NLP), although Recurrent Neural Networks which leverage Long Short Term Memory (LSTM) cells are more promising than CNN as they take into account the proximity of words to create trainable patterns.



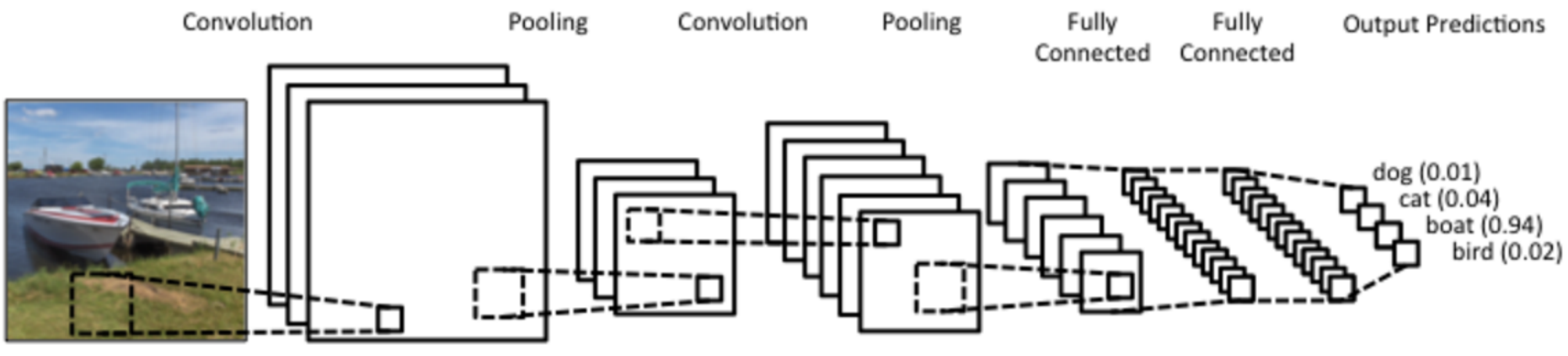

```
# deeper model with 3 layers
X_train = np.squeeze(X_train)
X_train = np.expand_dims(X_train,2)

X_test = np.squeeze(X_test)
X_test = np.expand_dims(X_test,2)

K.clear_session()
model = Sequential()
model.add(Input(shape=(8,1)))
model.add(Conv1D(filters = 24, kernel_size = 3, activation='relu'))
model.add(BatchNormalization())
model.add(Conv1D(filters = 48, kernel_size = 3, activation='relu'))
model.add(BatchNormalization())
model.add(Flatten())
model.add(Dense(units = 64, activation='relu'))
model.add(BatchNormalization())
model.add(Dense(2, activation='linear'))

model.build()
model.summary()

# compile the keras model
model.compile(loss='mse', optimizer='adam')

# fit the keras model on the dataset
history = model.fit(X_train, Y_train, epochs=E, batch_size=BS,validation_data=(X_test,Y_test),verbose=0)
plot_history(history)
print('Minimum Validation Loss: ',min(history.history['val_loss']))
y=model.predict(X_test[0,None,:])
print(y[0])
```

```
32 print(Y_test[0])
```

Deep Neural Network Architectures

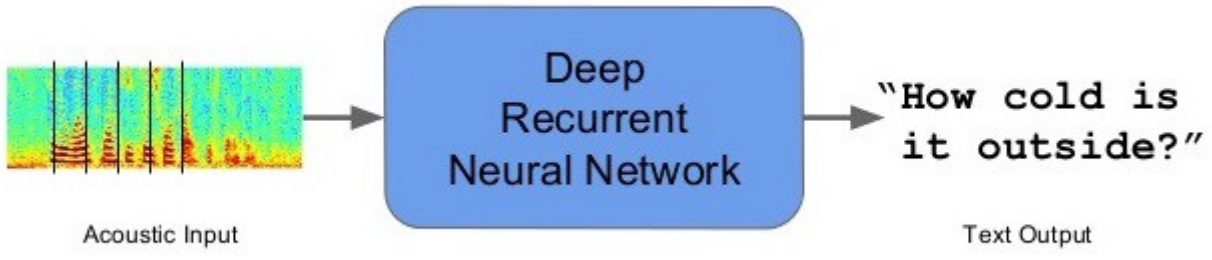
The most basic type of neural network is a multi-layer perceptron network, the type discussed above in the handwritten figures example, where data is fed forward between layers of neurons. Each neuron will typically transform the values they are fed using an activation function, which changes those values into a form that, at the end of the training cycle, will allow the network to calculate how far off it is from making an accurate prediction.

Convolutional Neural Networks for Image Understanding



There are various types of deep neural network, with structures suited to different types of tasks. For example, Convolutional Neural Networks (CNNs) are typically used for computer vision tasks, while Recurrent Neural Networks (RNNs) are commonly used for processing language. Each has its own specializations, in CNNs the initial layers are specialized for extracting distinct features from the image, which are then fed into a more conventional neural network to allow the image to be classified.

Recurrent Neural Networks and Language processing



RNNs differ from a traditional feed-forward neural network in that they don't just feed data from one neural layer to the next but also have built-in feedback loops, where data output from one layer is passed back to the layer preceding it—lending the network a form of memory. There is a more specialized form of RNN that includes what is called a memory cell and that is tailored to processing data with lags between inputs.

Generative Models

More recently, generative adversarial networks (GANS) are extending what is possible using neural networks. In this architecture two neural networks do battle, the generator network tries to create convincing "fake" data and the discriminator attempts to tell the difference between fake and real data. With each training cycle, the generator gets better at producing fake data and the discriminator gains a sharper eye for spotting those fakes. By pitting the two networks against each other during training, both can achieve better performance. GANs have been used to carry out some remarkable tasks, such as turning these dashcam videos from day to night or from winter to summer, as shown in the video below, and have applications ranging from turning low-resolution photos into high-resolution alternatives and generating images from written text. GANs have their own limitations, however, that can make them challenging to work with, although these are being tackled by developing more robust GAN variants.

Should you use always deep learning instead of machine learning?

No, because deep learning can be very expensive from a computational point of view. For non-trivial tasks, training a deep-neural network will often require processing large amounts of data using clusters of high-end GPUs for many, many hours.

If the problem can be solved using a simpler machine-learning algorithm such as Bayesian inference or linear regression, one that doesn't require the system to grapple with a complex combination of hierarchical features in the data, then these far less computational demanding options will be the better choice.

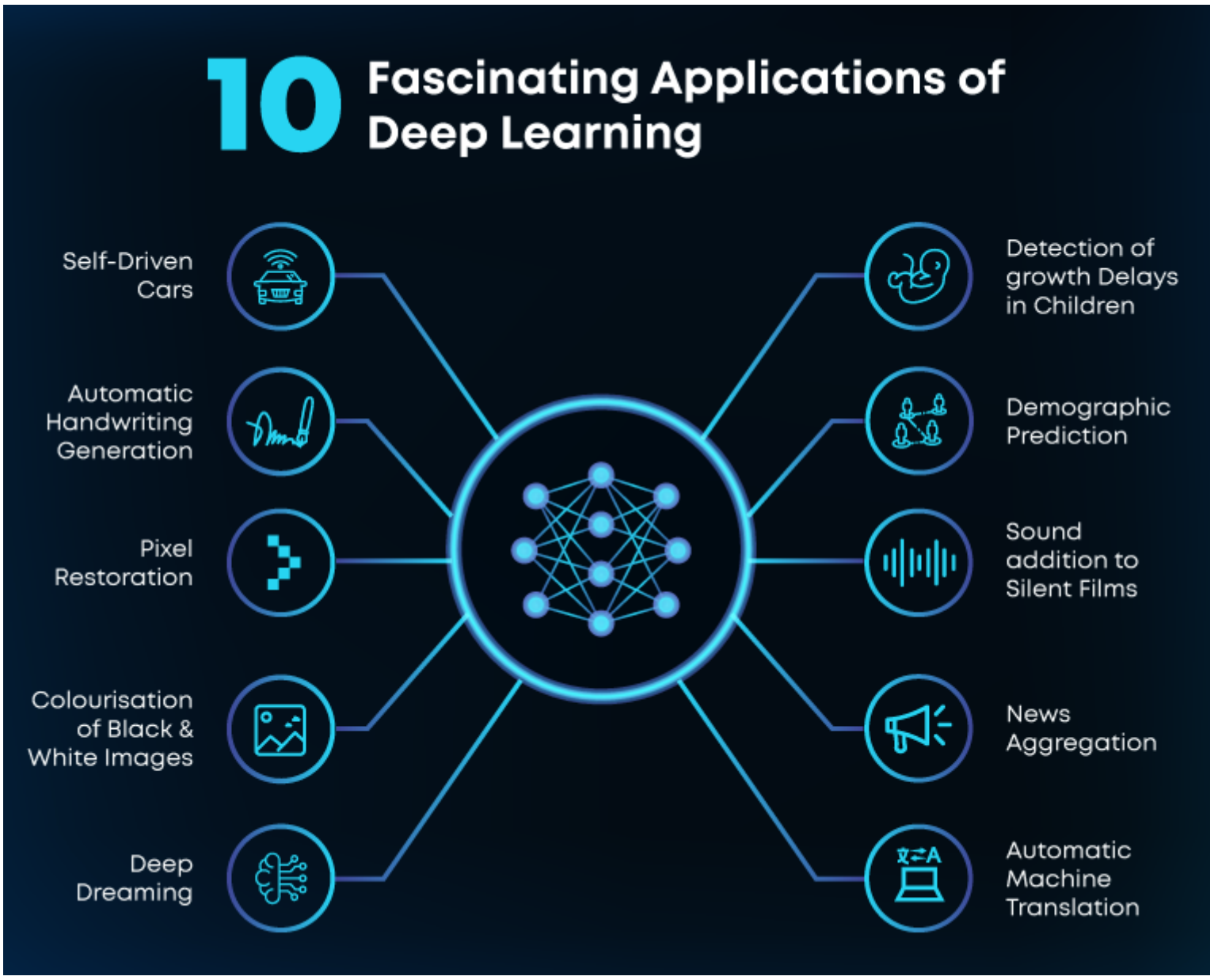
Deep learning may also not be the best choice for making a prediction based on data. For example, if the dataset is small then sometimes simple linear machine-learning models may yield more accurate results—although some machine-learning specialists argue a properly trained deep-learning neural network can still perform well with small amounts of data.

What are the drawbacks of deep neural networks?

One of the big drawbacks is the amount of data they require to train, with Facebook recently announcing it had used one billion images to achieve record-breaking performance by an image-recognition system. When the datasets are this large, training systems also require access to vast amounts of distributed computing power. This is another issue of deep learning, the cost of training. Due to the size of datasets and number of training cycles that have to be run, training often requires access to high-powered and expensive computer hardware, typically high-end GPUs or GPU arrays. Whether you're building your own system or renting hardware from a cloud platform, neither option is likely to be cheap.

Deep-neural networks are also difficult to train, due to what is called the vanishing gradient problem, which can worsen the more layers there are in a neural network. As more layers are added the vanishing gradient problem can result in it taking an unfeasibly long time to train a neural network to a good level of accuracy.

Applications



- **Virtual Assistants:** Amazon Echo, Google Assistant, Alexa, and Siri are all exploiting deep learning capabilities to build a customized user experience for you. **They 'learn' to recognize your voice and accent and present you a secondary human experience through a machine by using deep neural networks imitating not just speech but also the tone of a human.** Virtual assistants help you shop, navigate, take notes and translate them to text, and even make salon appointments for you.
- **Facial Recognition:** The iPhone's Facial Recognition uses deep learning to identify data points from your face to unlock your phone or spot you in images. Deep Learning helps them protect the phone from unwanted unlocks and making your experience hassle-free even when you have changed your hairstyle, lost weight, or in poor lighting. **Every time you unlock your phone, deep learning uses thousands of data points to create a depth map of your face and the inbuilt algorithm uses those to identify if it is really you or not.**
- **Personalization:** E-Commerce and Entertainment giants like Amazon and Netflix, etc. are building their deep learning capacities further to provide you with a personalized shopping or entertainment system. **Recommended items/series/movies based on your 'pattern' are all based on deep learning.** Their businesses thrive on pushing out options in your subconscious based on your preferences, recently visited items, affinity to brands/actors/artists, and overall browsing history on their platforms.
- **Natural Language Processing:** One of the most critical technologies, Natural Language Processing is taking AI from good to great in terms of use, maturity, and sophistication. Organizations are using deep learning extensively to enhance these complexities in NLP applications. **Document summarization, question answering, language modelling, text classification, sentiment analysis** are some of the popular applications that are already picking up momentum. Several jobs worldwide that depend on human intervention for verbal and written language expertise will become redundant as NLP matures.
- **Healthcare:** Another sector to have seen tremendous growth and transformation is the healthcare sector. From personal virtual assistants to **fitness bands and gears, computers are recording a lot of data about a person's physiological and mental condition every second.** Early detection of diseases and conditions, quantitative imaging, robotic surgeries, and availability of decision-support tools for professionals are turning out to be game-changers in the life sciences, healthcare, and medicine domain.
- **Autonomous Cars:** Uber AI Labs in Pittsburg are engaging in some tremendous work to make autonomous cars a reality for the world. Deep Learning, of course, is the guiding principle behind this initiative for all automotive giants. Trials are on with several autonomous cars that are learning better with more and more exposure. **Deep learning enables a driverless car to navigate by exposing it to millions of scenarios to make it a safe and comfortable ride. Data from sensors, GPS, geo-mapping is all combined together in deep learning** *to create models that specialize in identifying paths, street signs, dynamic elements like traffic, congestion, and pedestrians.
- **Text Generation:** Soon, deep learning will create original text (even poetry), as technologies for text generation is evolving fast. Everything from the large dataset comprising text from the internet to Shakespeare is being fed to **deep learning models to learn and emulate human creativity with perfect spelling, punctuation, grammar, style, and tone**. It is already generating caption/title on a lot of platforms which is testimony to what lies ahead in the future.
- **Visual Recognition:** Convolutional Neural Networks enable digital image processing that can further be segregated into facial recognition, object recognition, handwriting analysis, etc. Computers can now recognize images using deep learning. Image recognition technology refers to the technology that is based on the digital image processing technology and utilizes artificial intelligence technology, especially

the machine learning method, to make **computers recognize the content in the image. *_Further applications include_ *colouring black and white images and adding sound to silent movies** which has been a very ambitious feat for data scientists and experts in the domain.

https://www.mygreatlearning.com/blog/what-is-deep-learning/#six

https://images.nvidia.com/content/tegra/automotive/images/2016/solutions/pdf/end-to-end-dl-using-px.pdf