Computer vision and machine learning for the material scientist

Lecture 7. *Deep Neural Networks*

Henry Proudhon

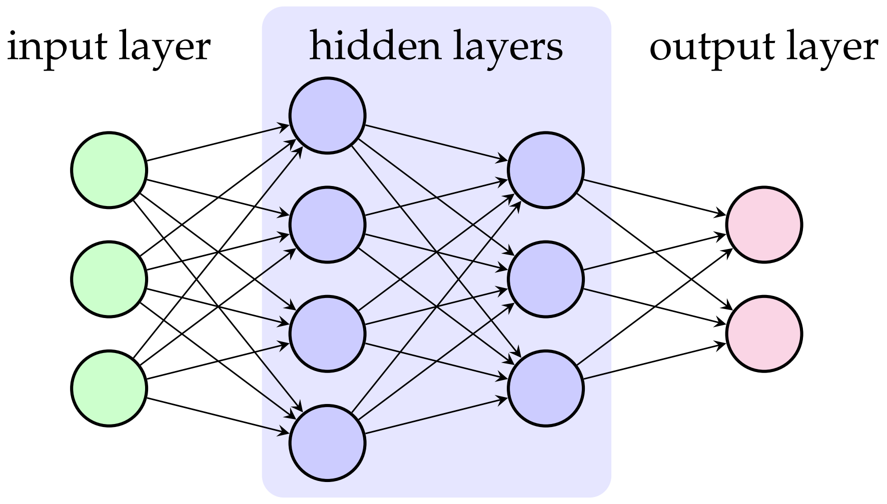
MINES ParisTech, PSL University
Centre des Matériaux, CNRS UMR 7633, Evry, France

@ Centre des Matériaux
February 25, 2021

Outline

Contents

Recall : Feed-Forward Neural Network



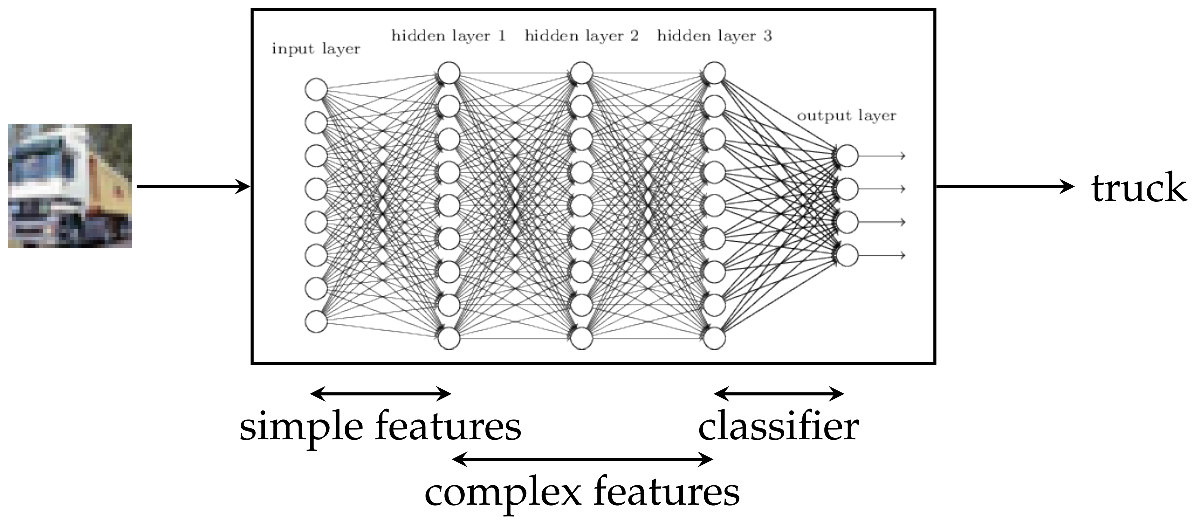
Training phase, offline, slow
- SGD : iterate on each training data point
- Forward pass : compute the network output
- Backprop pass : compute $\nabla_W L$ using the chain rule
- Update rule : modify the weigths W

Prediction phase, online, fast
- Predict the network output for a new unseen data point

End-to-end learning

Classical CV Image feature (human) engineered are fed to a machine learning classifier.

Deep learnig Image features and classification are learned at the same time !



How deep is deep ?

No clear answer to this question !

As a rule of thumb : a network with more than 2 hidden layer can be considered as deep. Before the 90s :

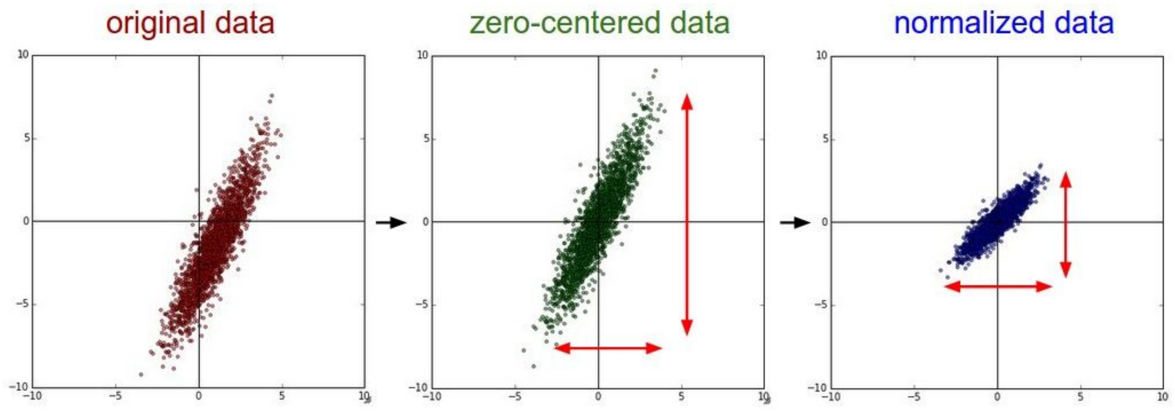
- data sets were too small
- computers were too slow
- initialization was bad
- activation functions were not the good ones

Today we have :

1. Faster computers
2. Highly optimized hardware (GPUs)
3. Large, labeled datasets in the order of millions of images (for some problems)
4. A better understanding of weight initialization
5. Superior activation functions

Data preprocessing

Network convergence can be very sensitive to how your features are distributed. So it is a good idea to **remove the mean** (make it zero-centered) and **normalize** it (variance equals 1).



In Numpy (X is the matrix of the data, X[i] a data point) :

```
X -= np.mean(X, axis=0)
X /= np.std(X, axis=0)
```

Weights initialization

Before we can train the network, we need to initialize all W.

Biases initialisation it is common to *initialize all biases to zero*.

Constant initialisation (eg 0.) is bad since each neuron will have the same input and therefore the same output and same gradients. Training will fail, we need to *break the symmetry* of the network.

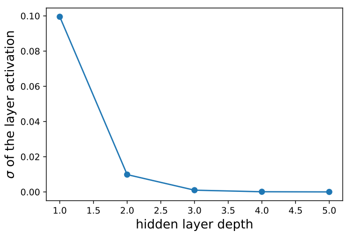
Random initialisation will work for shallow networks. Gradients scale with W so starting with very small values is not advised. Random initialization breaks down for deep networks.

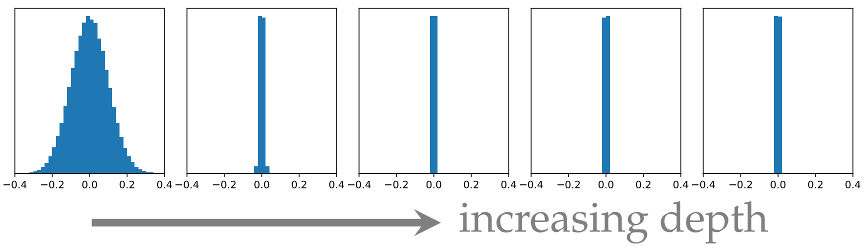
```
W = np.random.randn(D, H) * 0.01
```

Note that **batch normalisation** tends to reduce significantly initialisation problems.

Xavier initialization

Random initialisation quickly break down for deep networks. The reason for this is that the activation will quickly falls to zero after a few layers (and so does the gradients).





A better solution :

```
W = np.random.randn(D, H) / np.sqrt(D)
```

This is called Xavier initialisation [Glorot and Bengio, 2010] and ensure the activation on the output layer is non-zero.

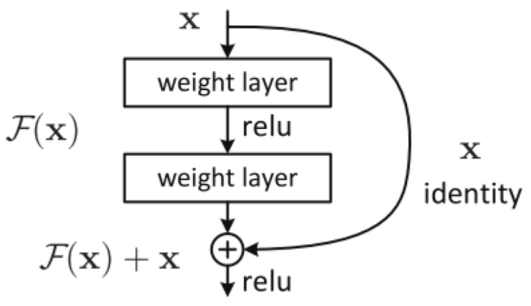
For ReLU activation functions [He et al., 2015b] :

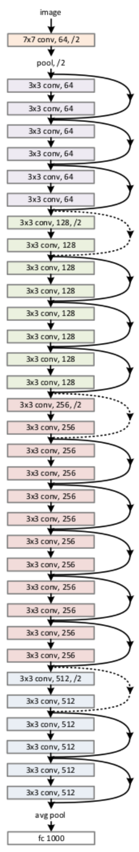
```
W = np.random.randn(D, H) * np.sqrt(2 / D)
```

Residual connections

Very deep neural networks suffer from vanishing gradients. To alleviate this problem, **residual connections** were proposed [He et al., 2015a].

- stacking layers should not degrade the performance (think identity)
- fit a residual mapping rather than the complete transformation
- since its introduction in 2015, the ResNet architecture has become very popular

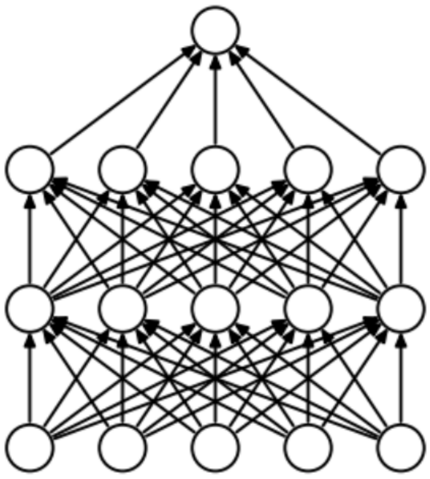


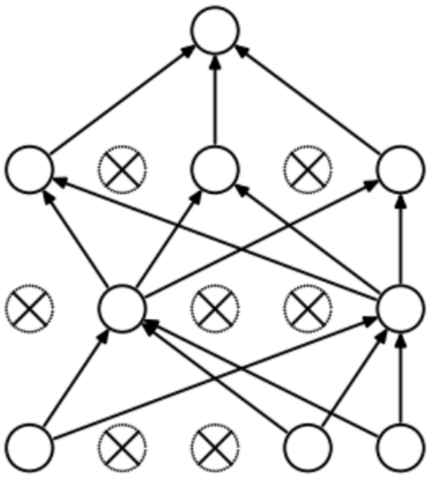


ResNet architecture

Dropout

Dropout is a form of regularization and is used to help the network to generalize better [Srivastava et al., 2014].


(a) Standard Neural Net


(b) After applying dropout.

Training ignore (zero out) a random fraction p of nodes (and corresponding activations)

Testing use all activations, but reduce them by a factor p to account for the missing activations during training

Recall from last lecture : single layer Neural Network

Efficient vectorized code using numpy.

```
import numpy as np

X, y = ...
w = np.random.randn(3, 1)  # first layer
eta = 1e-2  # learning rate
n_epochs = 100

for t in range(n_epochs):
    # forward pass
    y_pred = 1 + np.exp(-X.dot(w)  # activation
    loss = np.square(y_pred - y).sum()

    # backprop
    grad_y_pred = 2. * (y_pred - y)
    grad_W = X.T.dot(grad_y_pred * y_pred * (1 - y_pred))

    # update rule
    w -= eta * grad_W
```

A 2 layer Neural Network in 15 lines of Python

Efficient vectorized code using numpy.

```
import numpy as np

X, y = ...
w1 = np.random.randn(3, 3)  # first layer
w2 = np.random.randn(3, 1)  # second layer
eta = 1e-2  # learning rate
n_epochs = 10000

for t in range(n_epochs):
    # forward pass
    h = 1 / (1 + np.exp(-X.dot(w1)))  # first layer activation
    y_pred = h.dot(w2)  # activation of the second layer
    loss = np.square(y_pred - y).sum()

    # backprop
    grad_y_pred = 2. * (y_pred - y)
    grad_w2 = h.T.dot(grad_y_pred)
    grad_h = grad_y_pred.dot(w2.T)
    grad_w1 = X.T.dot(grad_h * h * (1 - h))

    # update rule
    w1 -= eta * grad_w1
    w2 -= eta * grad_w2
```

Contents

Solving the XOR problem

Training phase :

```
# XOR data set
X = np.array([[1, 0, 0],
              [1, 0, 1],
              [1, 1, 0],
              [1, 1, 1]])
y = np.array([[0], [1], [1], [0]])

... perform learning
```

Prediction phase :

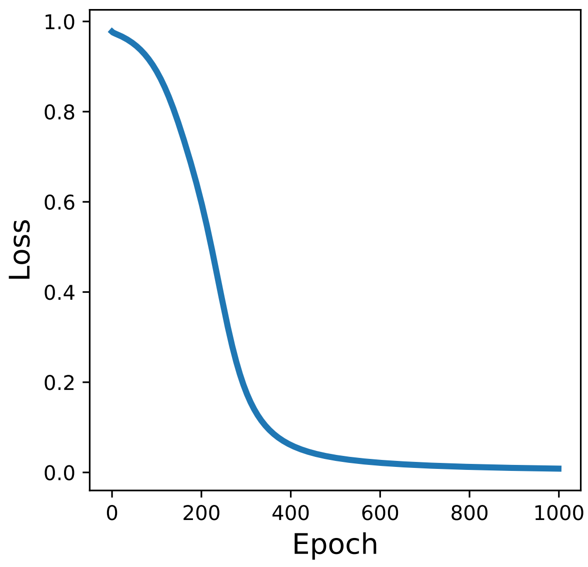
```
for (xi, yi) in zip(X, y):
    h = 1 / (1 + np.exp(-xi.dot(w1)))
    out = h.dot(w2)
    y_pred = 1 if out > 0.5 else 0
```

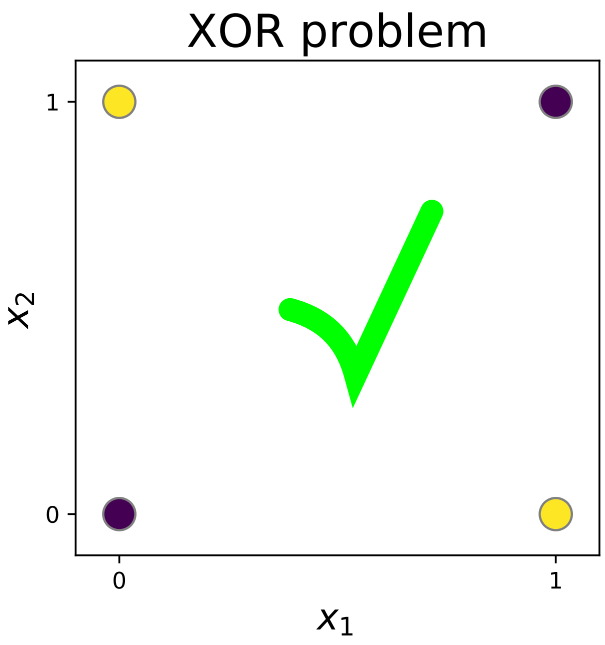
Output :

```
data=[1 0 0], ground-truth=[0], out=0.001, y=0
data=[1 0 1], ground-truth=[1], out=0.999, y=1
data=[1 1 0], ground-truth=[1], out=0.999, y=1
data=[1 1 1], ground-truth=[0], out=0.002, y=0
```

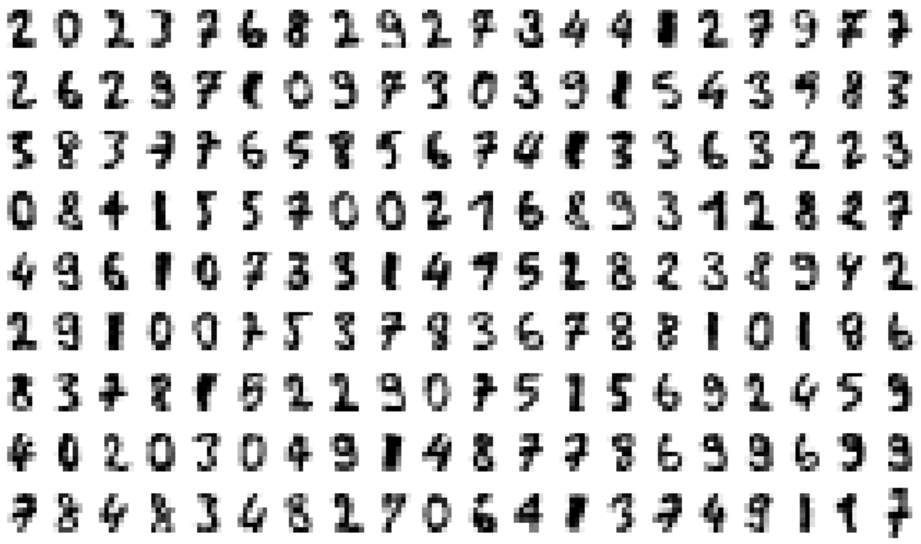
Results

The network correctly solve the XOR problem after $\sim$ 300 epochs.



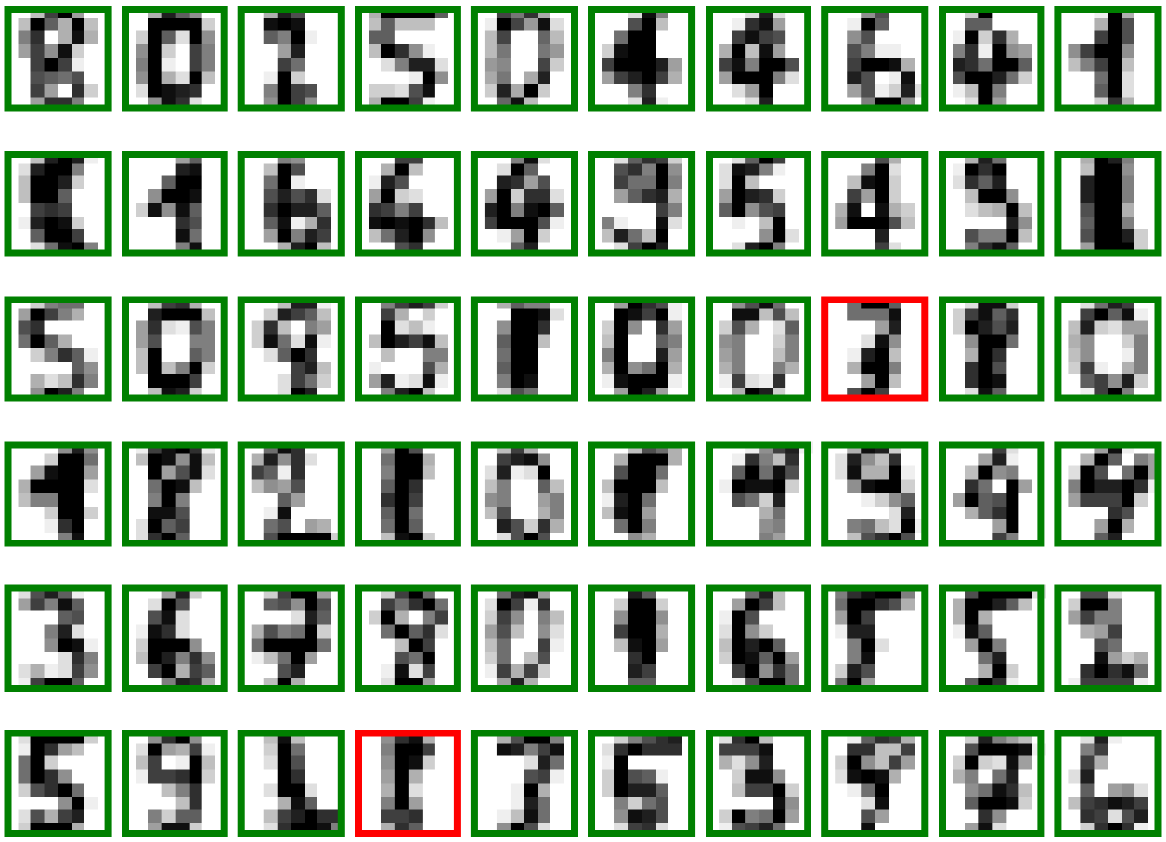


Training a deep neural network to tackle MNIST

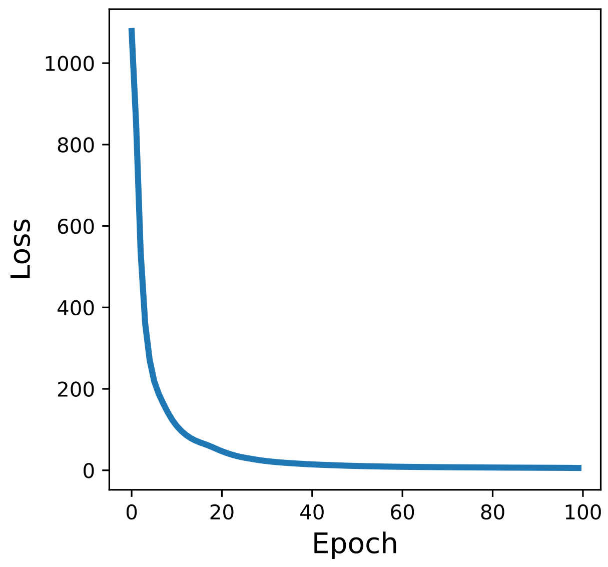


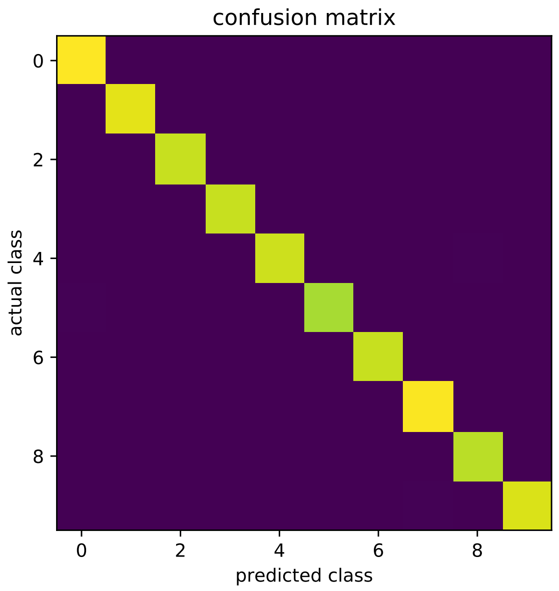
Results

The network easily achieves > 95% accuracy after 100 epochs.



Loss and confusion matrix

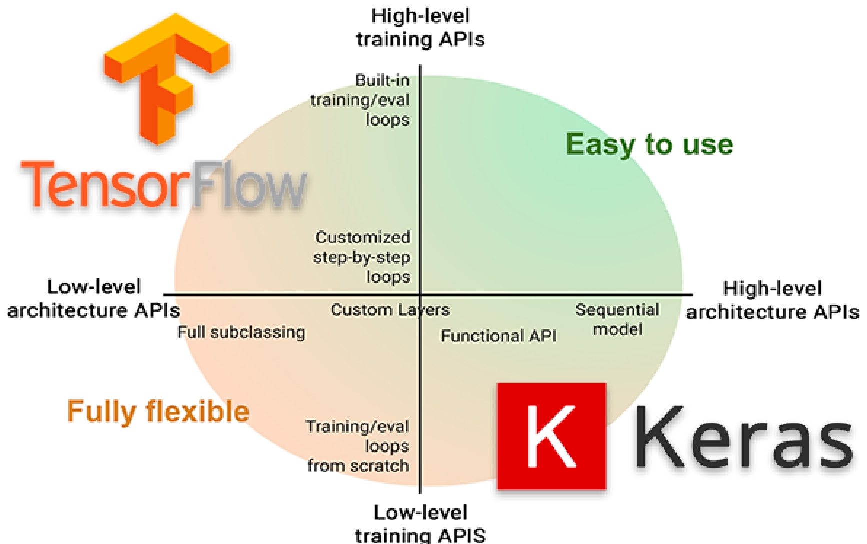




Keras and TensorFlow

Keras is a popular, dedicated, very easy to use library to build Deep Neural Networks.

Author	It was written by François Chollet at Google (since 2015) and is now part of TensorFlow.
Backend	Keras needs a backend, default is now TensorFlow (Theano until 1.1.0)
TensorFlow	Since 2019, Keras ships within TensorFlow and may be uses directly in Python with `tf.keras`



Solving Mnist in Python using Keras : model

```
X_train, y_train, X_test, y_test = ...

# define the 64-32-16-10 FC architecture using
    ↪ Keras
model = Sequential()
model.add(Dense(32, input_shape=(64,),
                    activation='sigmoid'))
model.add(Dense(16, activation='sigmoid'))
model.add(Dense(10, activation='softmax'))

# train the network
sgd = SGD(0.01)
model.compile(loss='categorical_crossentropy',
              optimizer=sgd, metrics=['accuracy'])
H = model.fit(X_train, y_train,
              validation_data=(X_test, y_test),
              epochs=200, batch_size=10)
```

Solving Mnist in Python using Keras : training

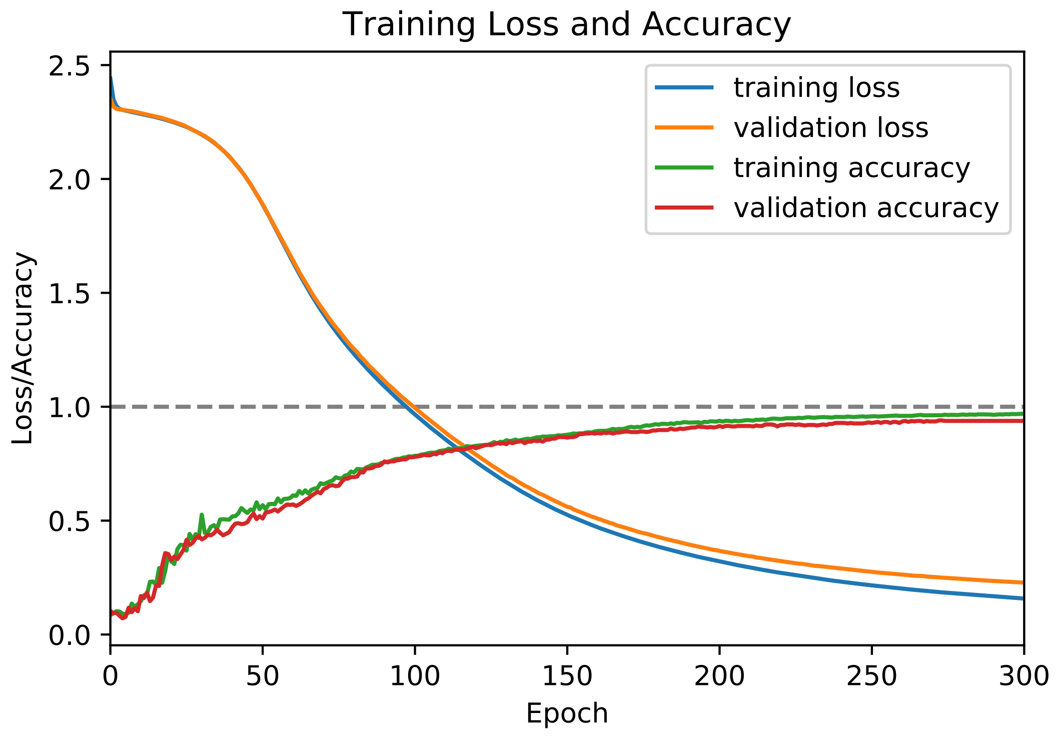
```
training network...
Train on 1347 samples, validate on 450 samples
Epoch 1/1000
1347/1347 [==============================] - 1s 916
    ↪ us/sample - loss: 2.4406 - accuracy: 0.0846
    ↪ - val_loss: 2.3521 - val_accuracy: 0.1044
Epoch 2/1000
1347/1347 [==============================] - 1s 501
    ↪ us/sample - loss: 2.3500 - accuracy: 0.0943
    ↪ - val_loss: 2.3172 - val_accuracy: 0.0933

...

Epoch 1000/1000
1347/1347 [==============================] - 1s 555
    ↪ us/sample - loss: 0.0261 - accuracy: 0.9993
    ↪ - val_loss: 0.1426 - val_accuracy: 0.9667
```

Results

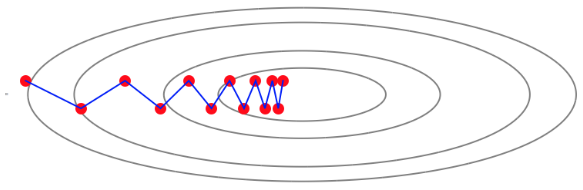
The network achieves > 97% accuracy after 300 epochs, 99.9% after 1000 epochs.

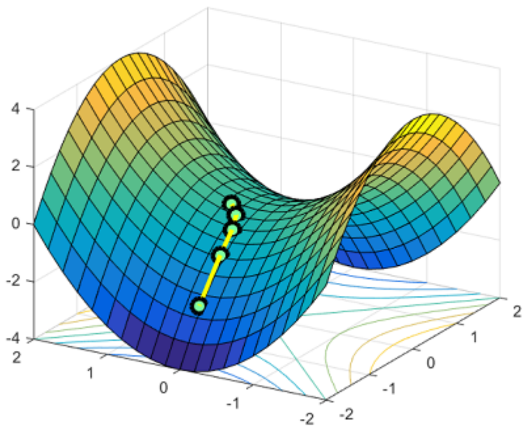


Problems with SGD

1. **Poor conditioning** (very different values of the gradient in different directions)
2. **Local minima and saddle points** (problem not convex)
3. **Noise in the gradient** from one iteration to the next (due to the stochastic character)

NB : All of these are **likely to happen** in very high dimensions.





Momentum

Recall the **vanilla SGD** :

$$W \leftarrow W - \eta \nabla_W L$$

SGD + Momentum :

$$v \leftarrow \rho v + \nabla_W L$$

$$W \leftarrow W - \eta v$$

- Build up velocity as a running mean during gradient descent (help escape local minima and saddle points)
- ρ is equivalent to some *friction* that slows down and decreases the momentum ($\rho = 0.9$)
- $v = 0$ at the beginning
- Very simple solution that kind of solves all 3 problems

Nesterov acceleration

Nesterov accelerated gradient (NAG) or Nesterov Momentum
Idea : step in the direction of the velocity, evaluate the gradient there and then update the weights.

$$v \leftarrow \rho v + \nabla_W L(W + \rho v)$$

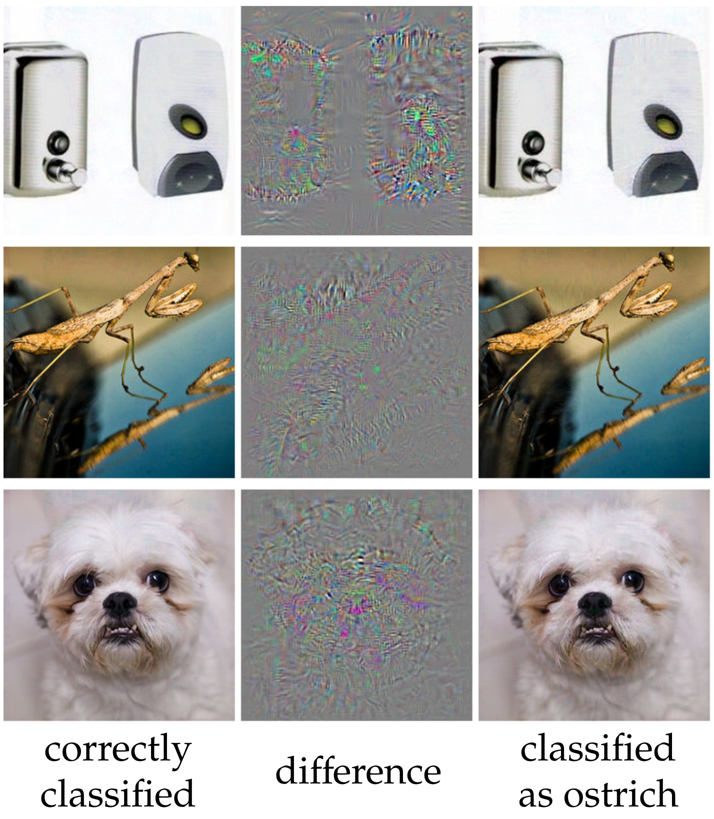
$$W \leftarrow W - \eta v$$

Act as a **corrective update** to the momentum method.

NB : many more method exist like Adagrad, RMSProp, Adam, . . .

Contents

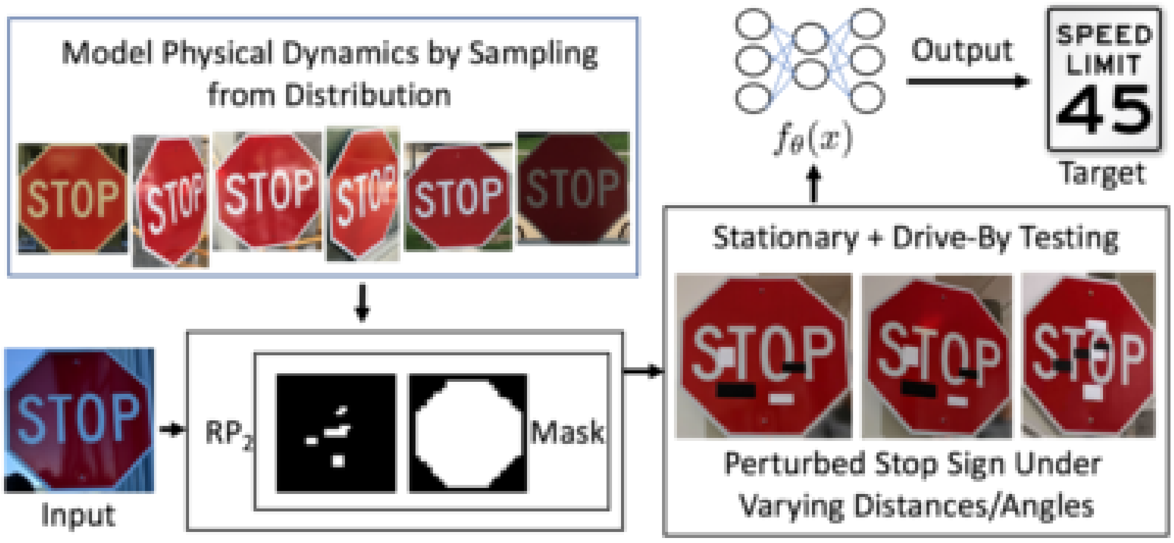
Advsersarial Attacks



Adversarial examples generated for AlexNet. All images are recognized as *ostrich* [Szegedy et al., 2013].

Robust Physical Perturbations attacks (RP_2)

Design realistic attacks (graffitis) on *stop signs* so they are classifued as a *Speed Limit 45 sign*.


Model Physical Dynamics by Sampling from Distribution
STOP
Input
RP2
Mask
Stationary + Drive-By Testing
Perturbed Stop Sign Under Varying Distances/Angles
$f_\theta(x)$
Output
SPEED LIMIT 45
Target

Misclassification in 100% of the images obtained in lab settings, and in 84.8% of the captured video frames obtained on a moving vehicle [Eykholt et al., 2018].

Out of context data points

Courtesy of Florida Highway Patrol
NTSB
Making Transportation Safer
YESTERDAY ★ TODAY ★ TOMORROW

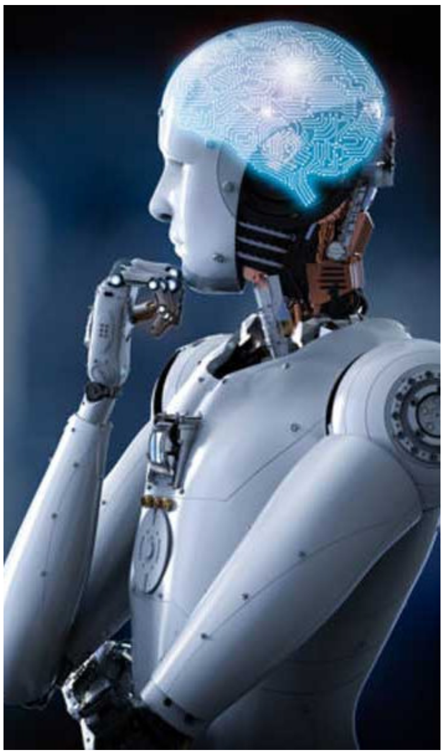
The system's cameras and radars did not detect the semi-truck that was crossing the driver's path before the fatal collision.

Validation of deep learning predictions

So far, **no formal validation proof** can be made for deep learning predictions. Lots of work being done on explainable AI. Are we ready to trade performance with explainability ?

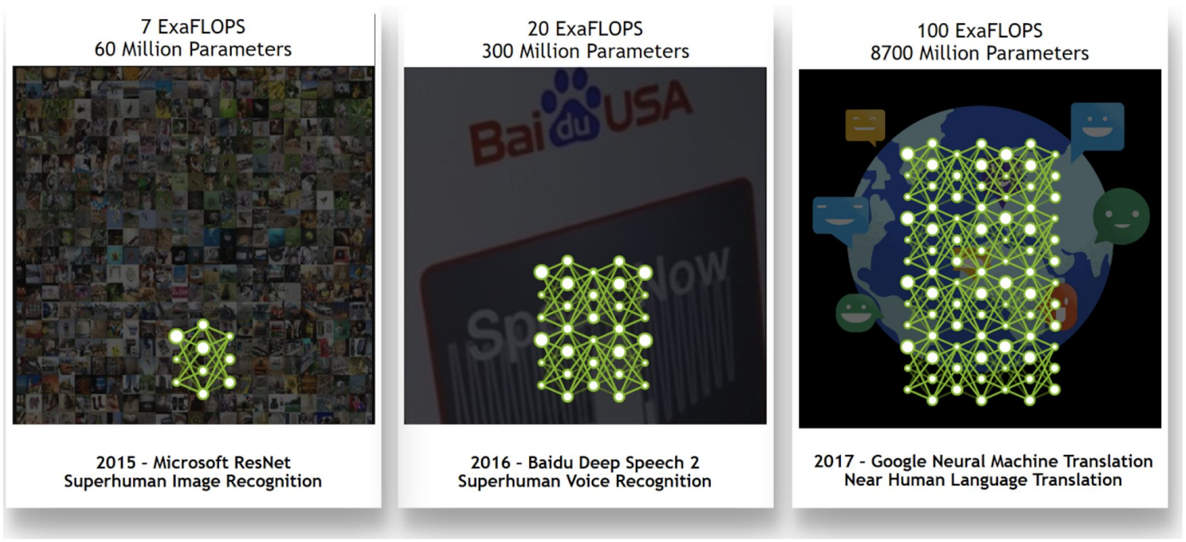
Mostly experimental using test data and cross validation (statistical validation). But then **how can we assess the coverage of possible inputs ?** This is a big open question.

Validation of the training data (avoid data poisoning) → Need for traceability. Another big question.



Other ethical issues in deep learning science

- A race to the largest network (essentially limited by your number of GPUs)
- Reproducibility highly questionable (but this is true for other sciences as well)
- Ecological cost :(


7 ExaFLOPS
60 Million Parameters
2015 - Microsoft ResNet
Superhuman Image Recognition
20 ExaFLOPS
300 Million Parameters
Baidu USA
2016 - Baidu Deep Speech 2
Superhuman Voice Recognition
100 ExaFLOPS
8700 Million Parameters
2017 - Google Neural Machine Translation
Near Human Language Translation

Contents

Summary

We have reviewed how deep learning extend the principles of machine learning and can achieve great performances.

- Large networks can be constructed easily and training with supervised learning using backprop
- Multi layered networks have no problem dealing with non-linearly separable data sets.
- State of the art tools are now available to the average user (such as the material scientist)
- Many hyper-parameters have still to be tuned by try-error / grid-search / expert knowledge.

Beside interesting results, when working with imaging data sets sensitive to translation, rotation, occlusion, intra-class variation, a special type of Feed-forward network is desirable : *Convolutional Neural Networks*.

Eykholt, K., Evtimov, I., Fernandes, E., Li, B., Rahmati, A., Xiao, C., Prakash, A., Kohno, T., and Song, D. (2018).
Robust physical-world attacks on deep learning models.

Glorot, X. and Bengio, Y. (2010).
Understanding the difficulty of training deep feedforward neural networks.
In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10)*. Society for Artificial Intelligence and Statistics.

He, K., Zhang, X., Ren, S., and Sun, J. (2015a).
Deep residual learning for image recognition.
CoRR, abs/1512.03385.

He, K., Zhang, X., Ren, S., and Sun, J. (2015b).
Delving deep into rectifiers : Surpassing human-level performance on imagenet classification.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014).
Dropout : A simple way to prevent neural networks from overfitting.
Journal of Machine Learning Research, 15 :1929–1958.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2013).
Intriguing properties of neural networks.