Deep Learning

A practical introduction

Emilio Sansano Sansano

esansano@uji.es

GIANT Research Group on Machine Learning for Smart Environments

init institute of new imaging technologies

UJI UNIVERSITAT JAUME I

GIANT RESEARCH GROUP

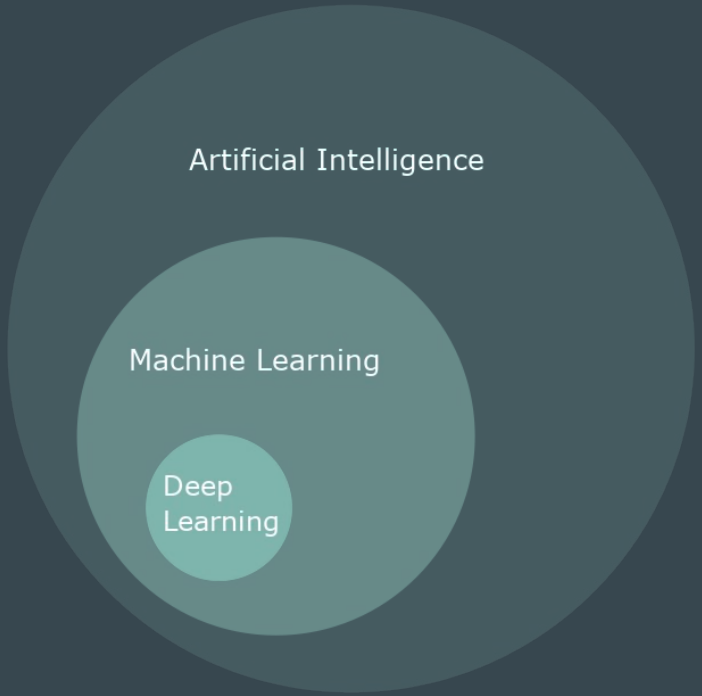
Research topics:

- Indoor Positioning
- Heathcare
- Human Activity Recognition
- Machine Learning on Video Games

What is Deep Learning?

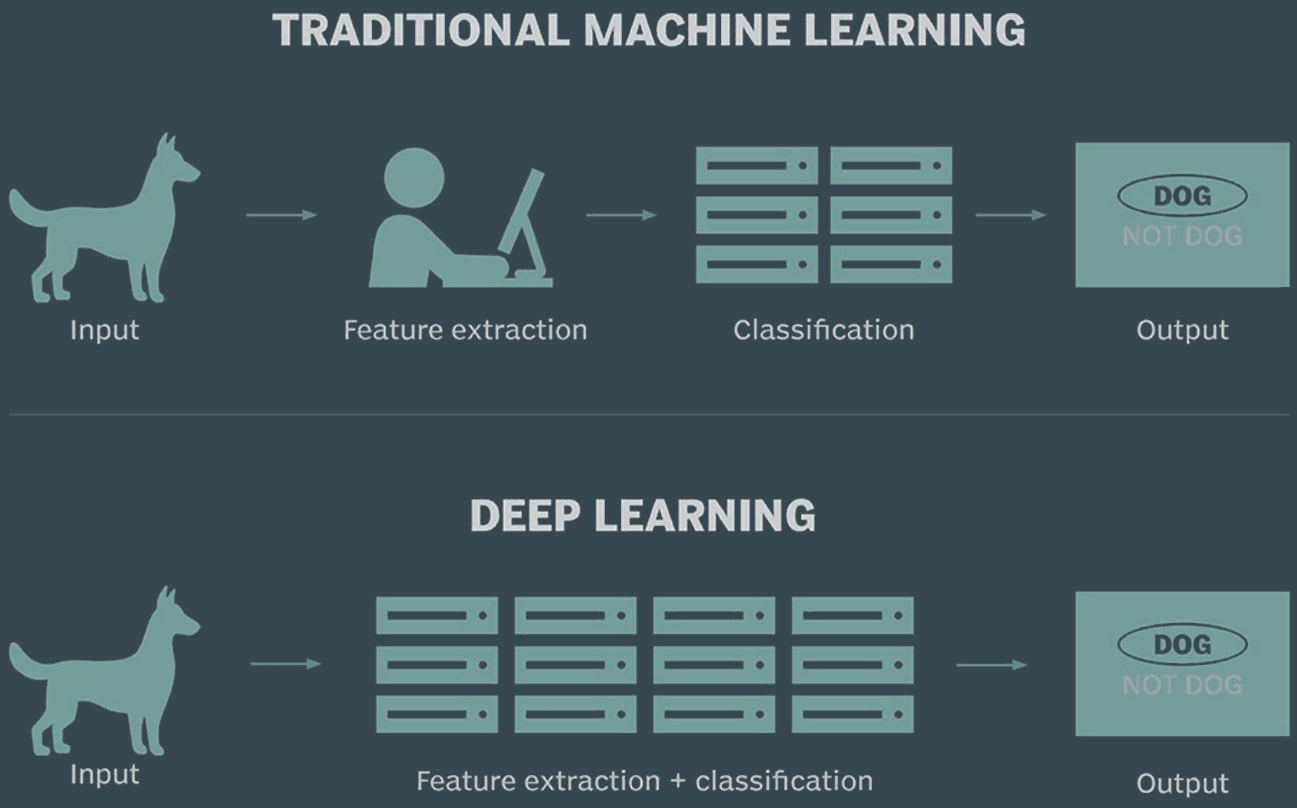
What is Deep Learning?

A subfield of machine learning.



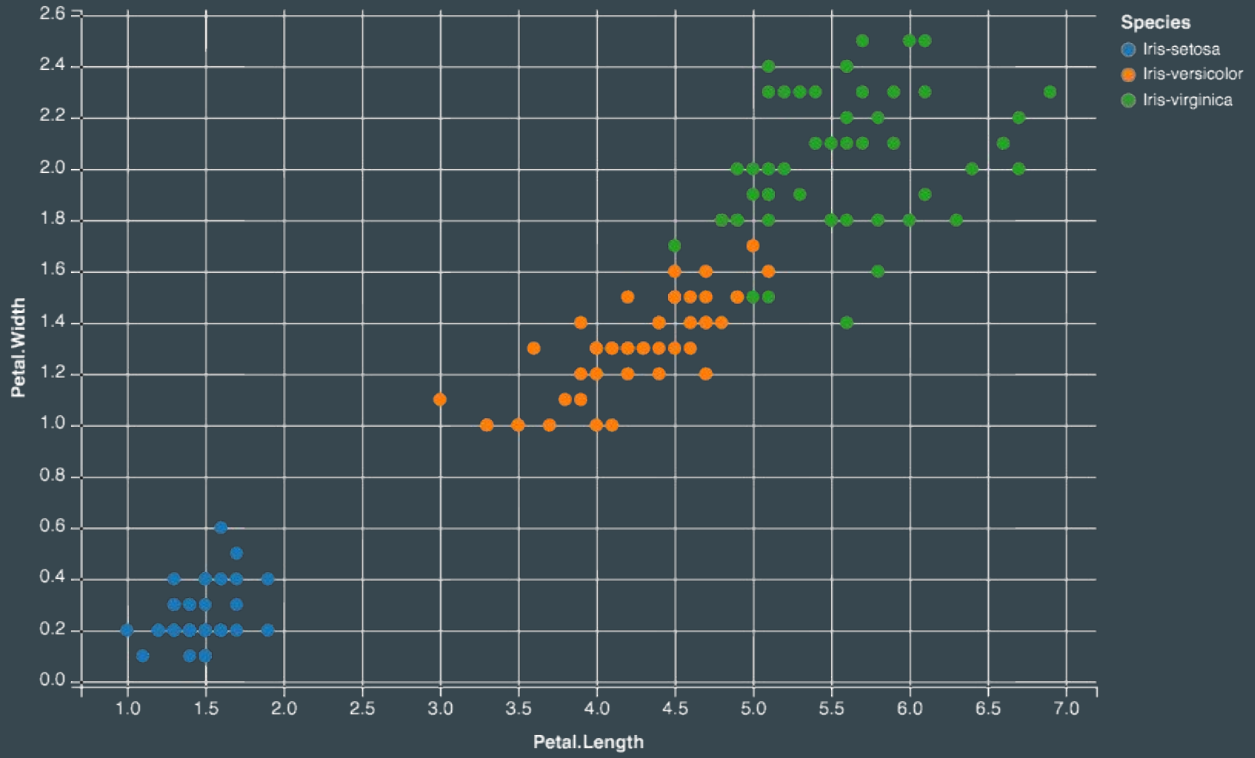
Inspired by the structure and function of the brain.

Deep Learning vs Traditional Machine Learning



Deep Learning. Automatic Feature Extraction.

Deep Learning. Automatic Feature Extraction.

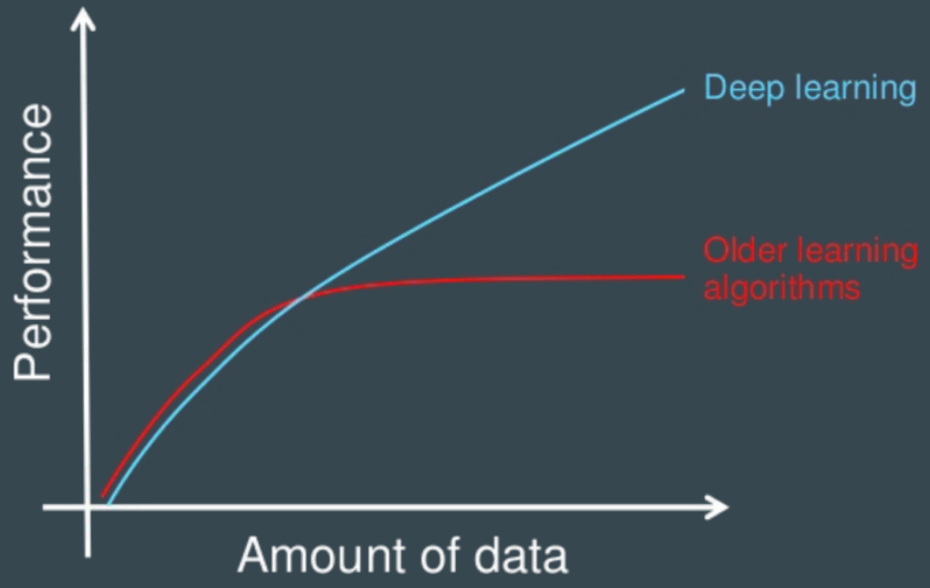


Iris Setosa

Iris Versicolor

Iris Virginica

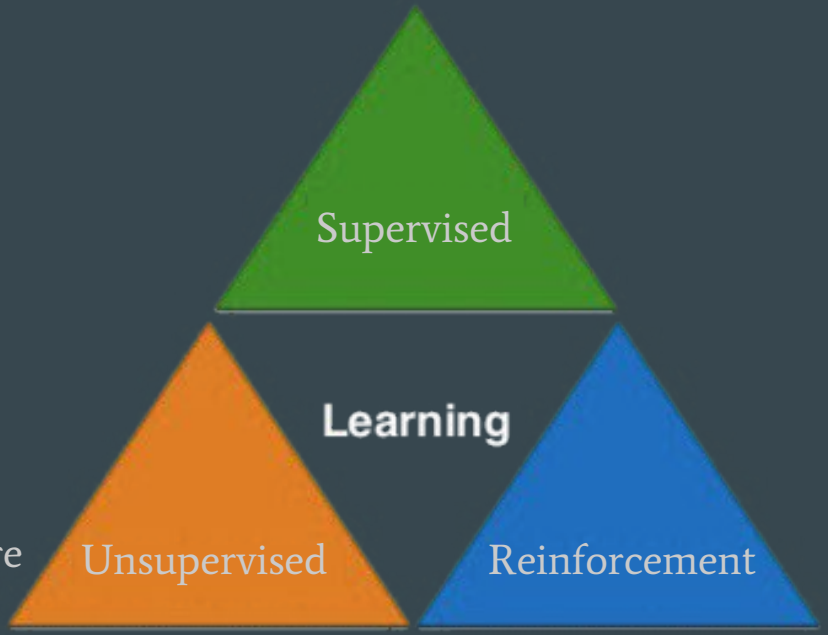
Deep Learning. Performance.



How do data science techniques scale with amount of data?

Deep Learning.

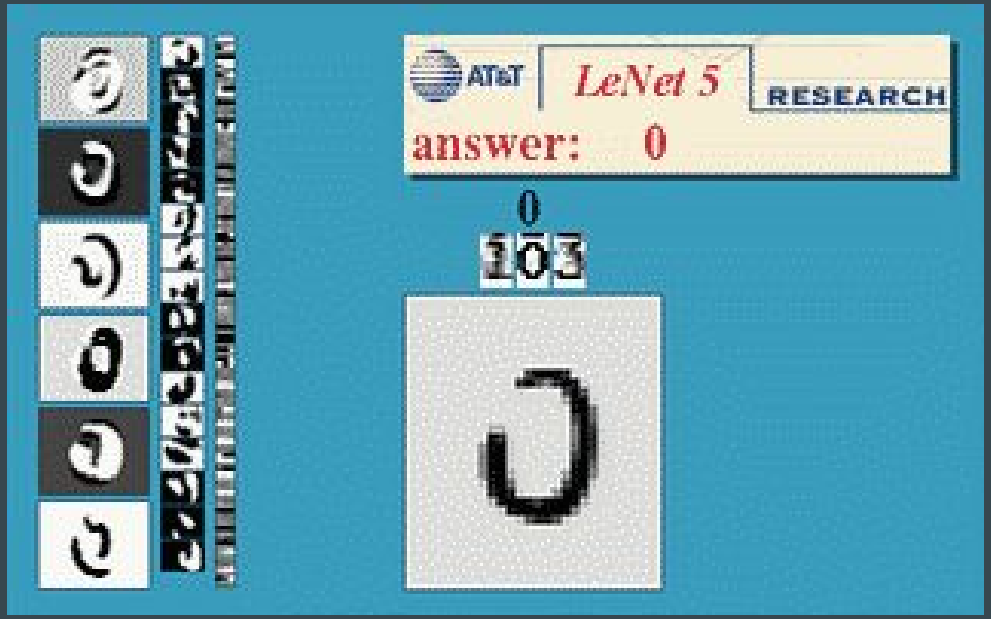
Labeled data
Direct feedback
Predict outcome/future



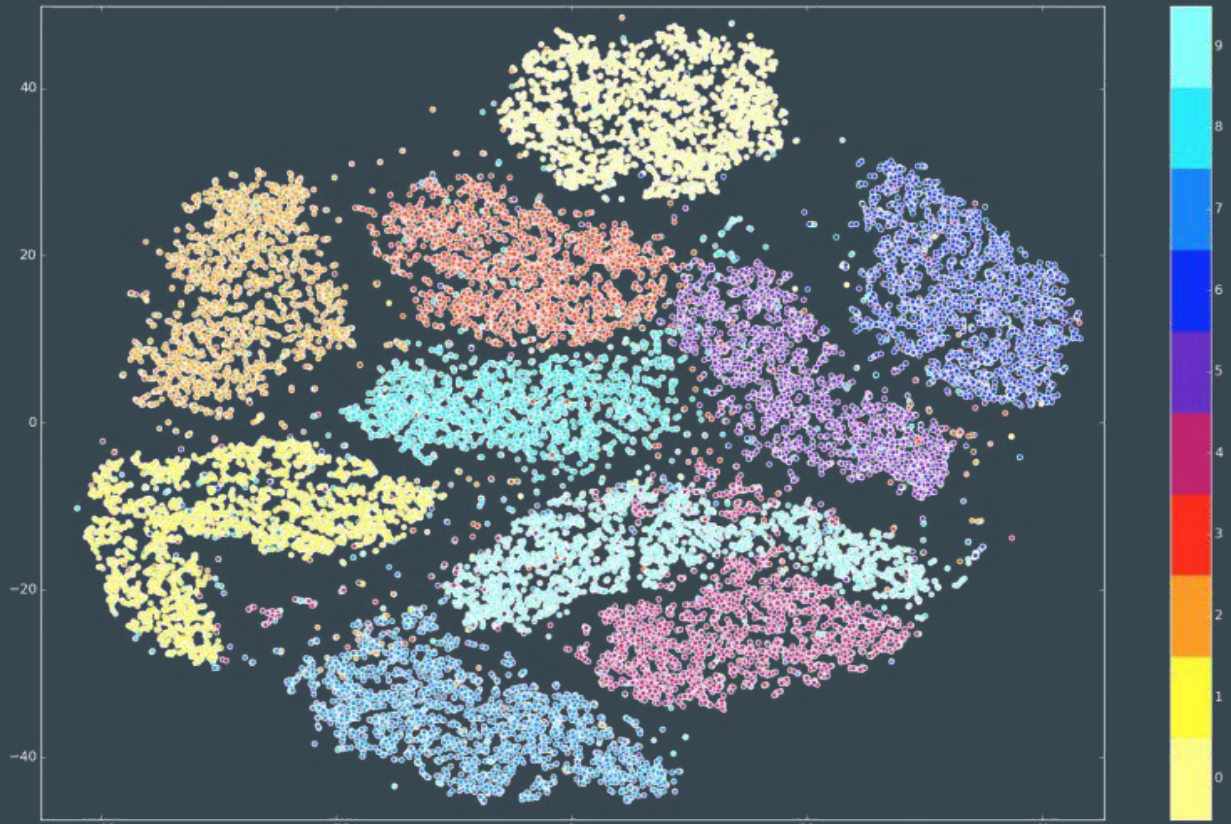
No labels
No feedback
Finds hidden structure

Decision process
Reward system
Learn series of actions

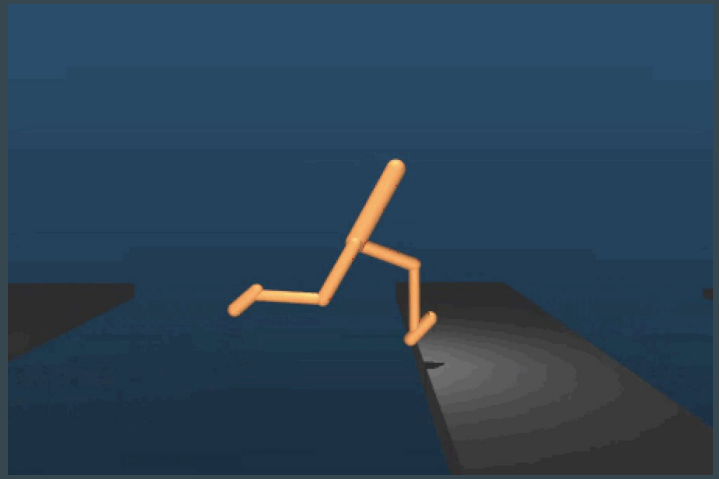
Supervised Deep Learning.

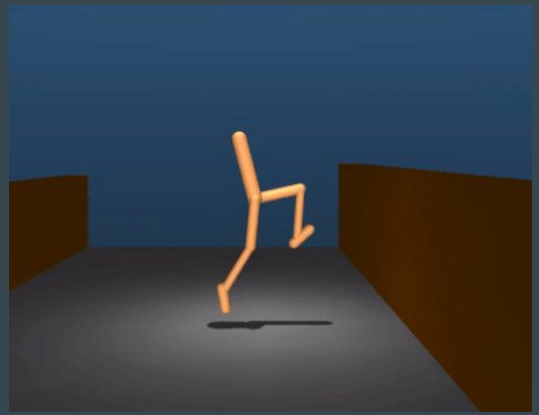


Unsupervised Deep Learning.

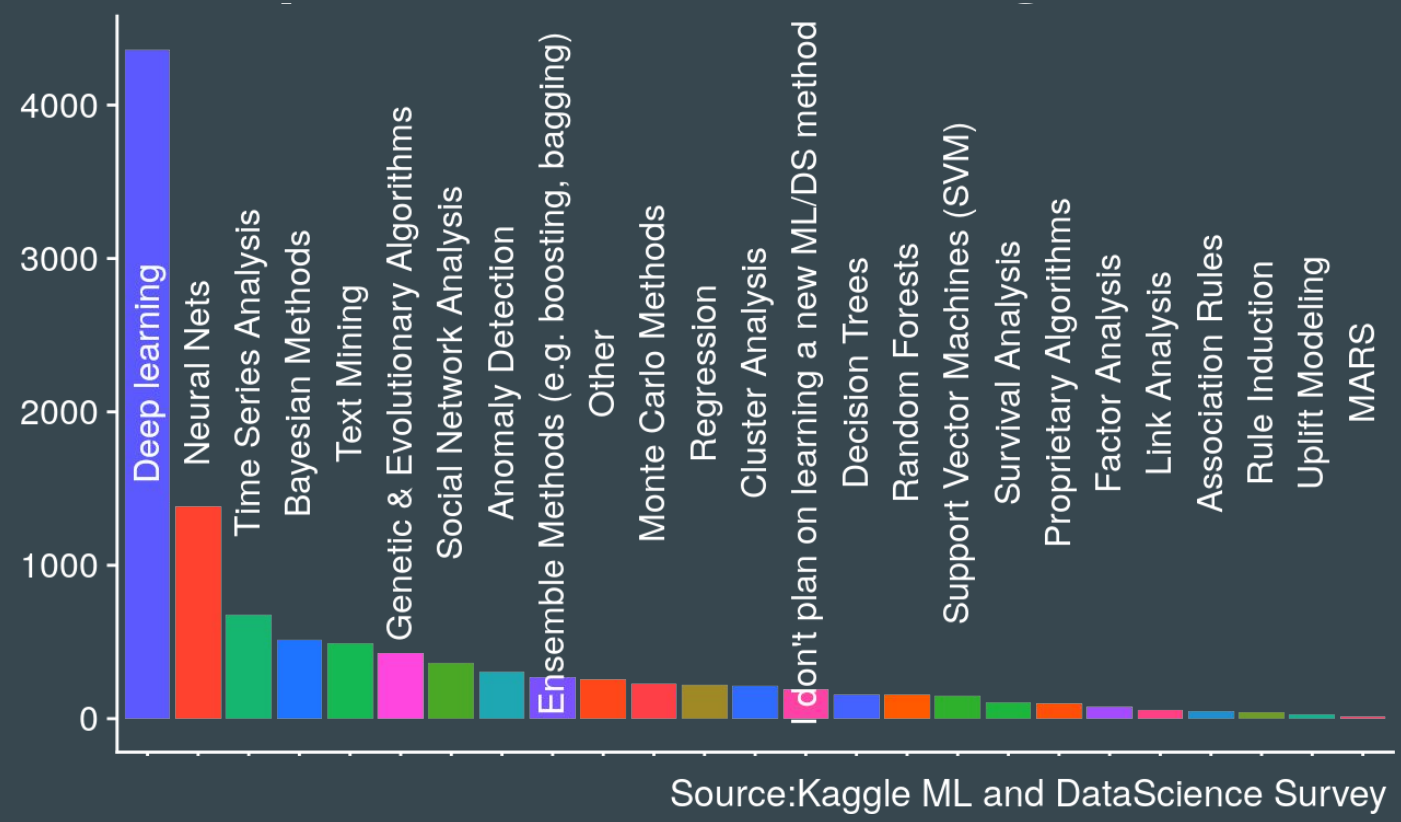


Reinforcement Deep Learning.

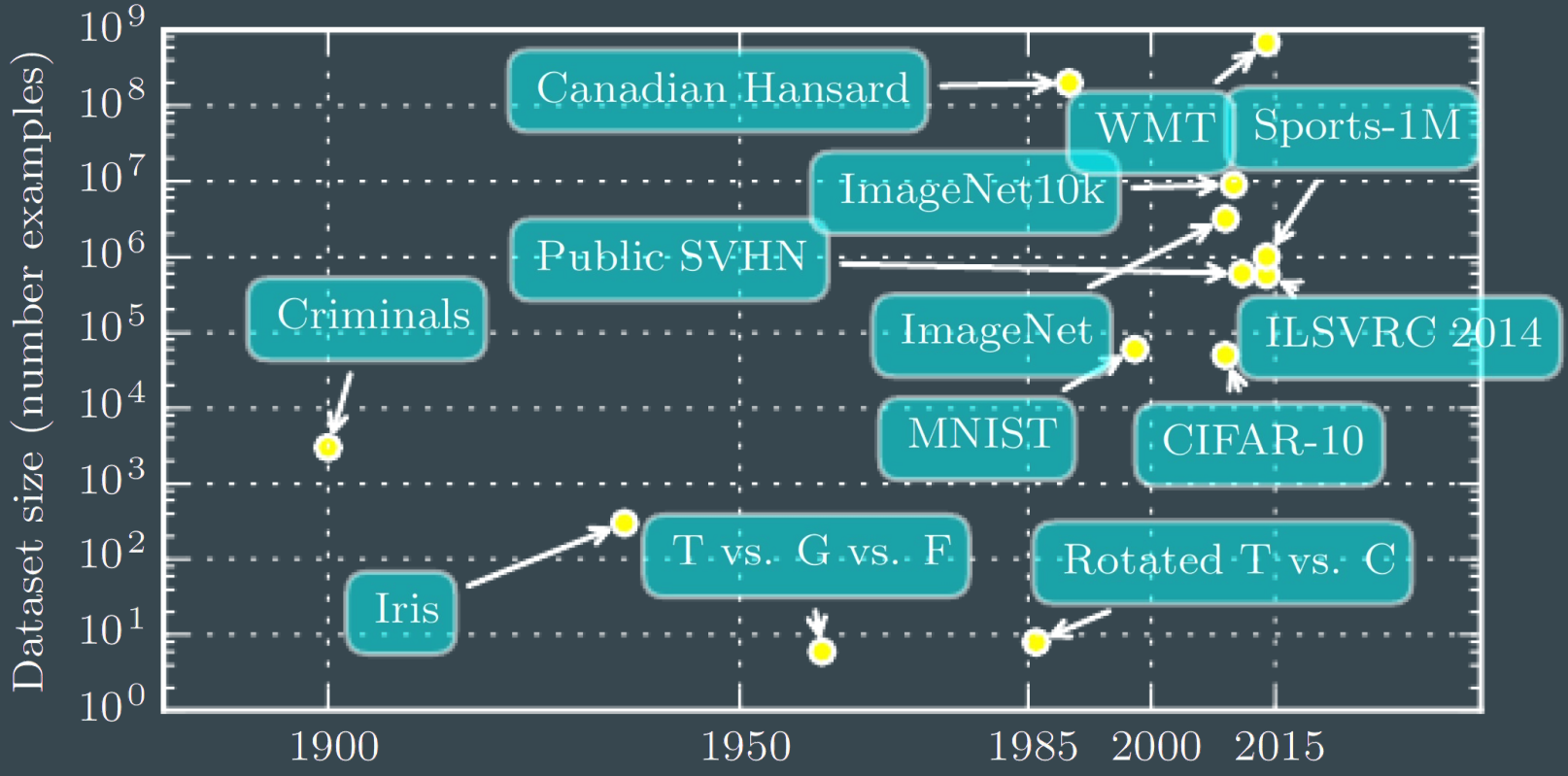




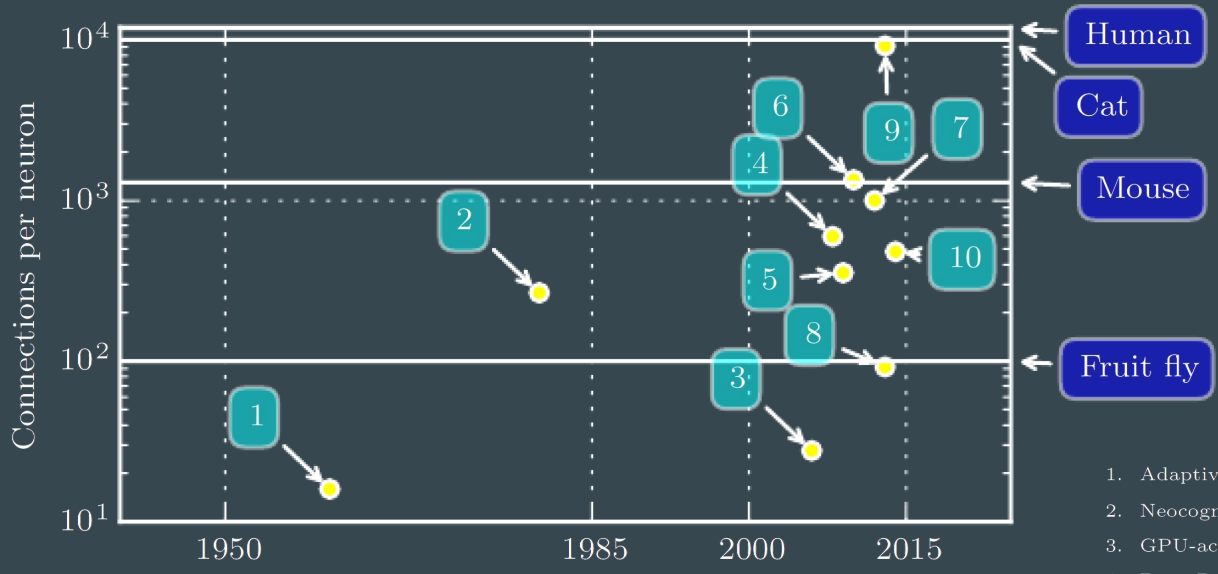
Popular ML Algorithms (kaggle).



Bigger data sets.

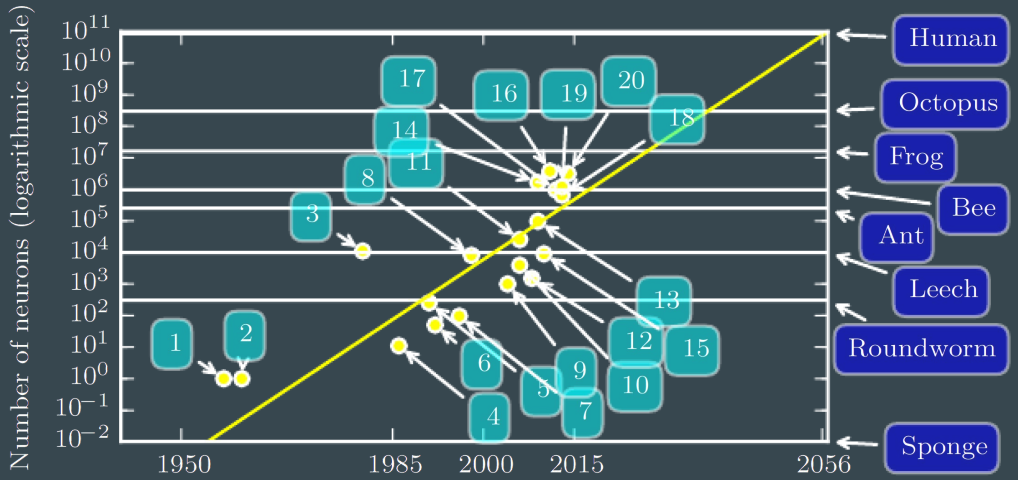


More connections per neuron.



1. Adaptive linear element (Widrow and Hoff, 1960)
2. Neocognitron (Fukushima, 1980)
3. GPU-accelerated convolutional network (Chellapilla *et al.*, 2006)
4. Deep Boltzmann machine (Salakhutdinov and Hinton, 2009a)
5. Unsupervised convolutional network (Jarrett *et al.*, 2009)
6. GPU-accelerated multilayer perceptron (Ciresan *et al.*, 2010)
7. Distributed autoencoder (Le *et al.*, 2012)
8. Multi-GPU convolutional network (Krizhevsky *et al.*, 2012)
9. COTS HPC unsupervised convolutional network (Coates *et al.*, 2013)
10. GoogLeNet (Szegedy *et al.*, 2014a)

More neurons.

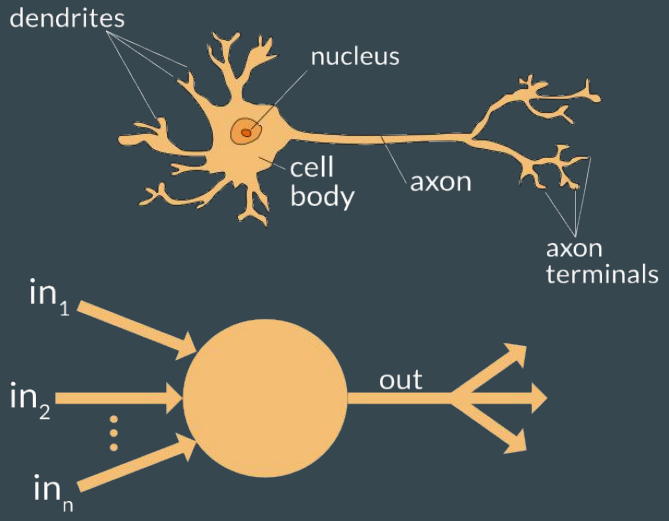


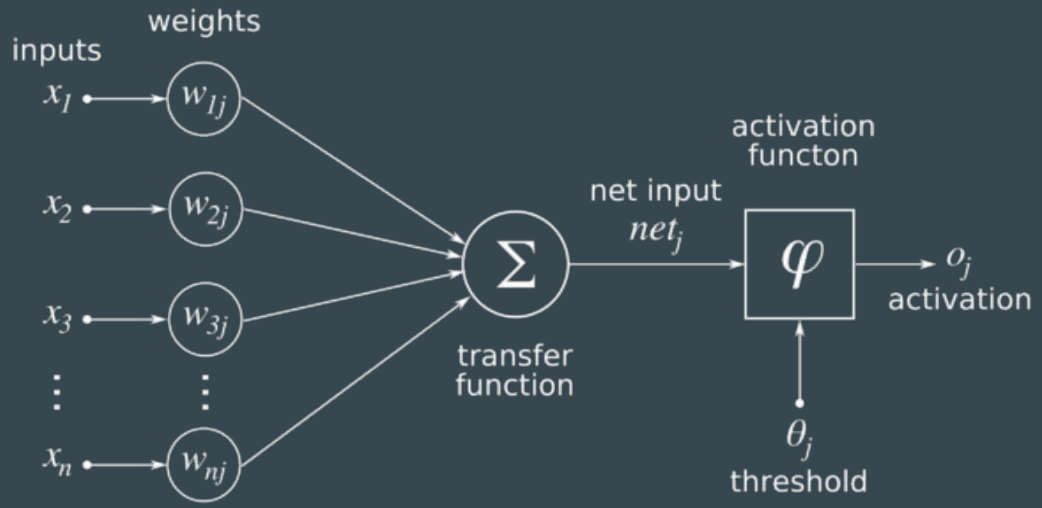
1. Perceptron (Rosenblatt, 1958, 1962)
2. Adaptive linear element (Widrow and Hoff, 1960)
3. Neocognitron (Fukushima, 1980)
4. Early back-propagation network (Rumelhart *et al.*, 1986b)
5. Recurrent neural network for speech recognition (Robinson and Fallside, 1991)
6. Multilayer perceptron for speech recognition (Bengio *et al.*, 1991)
7. Mean field sigmoid belief network (Saul *et al.*, 1996)
8. LeNet-5 (LeCun *et al.*, 1998b)
9. Echo state network (Jaeger and Haas, 2004)
10. Deep belief network (Hinton *et al.*, 2006)
11. GPU-accelerated convolutional network (Chellapilla *et al.*, 2006)
12. Deep Boltzmann machine (Salakhutdinov and Hinton, 2009a)
13. GPU-accelerated deep belief network (Raina *et al.*, 2009)
14. Unsupervised convolutional network (Jarrett *et al.*, 2009)
15. GPU-accelerated multilayer perceptron (Ciresan *et al.*, 2010)
16. OMP-1 network (Coates and Ng, 2011)
17. Distributed autoencoder (Le *et al.*, 2012)
18. Multi-GPU convolutional network (Krizhevsky *et al.*, 2012)
19. COTS HPC unsupervised convolutional network (Coates *et al.*, 2013)
20. GoogLeNet (Szegedy *et al.*, 2014a)

How does Deep Learning work?

Deep Learning. Conputational Neuron.

McCulloch & Pitts neuron model (1943)

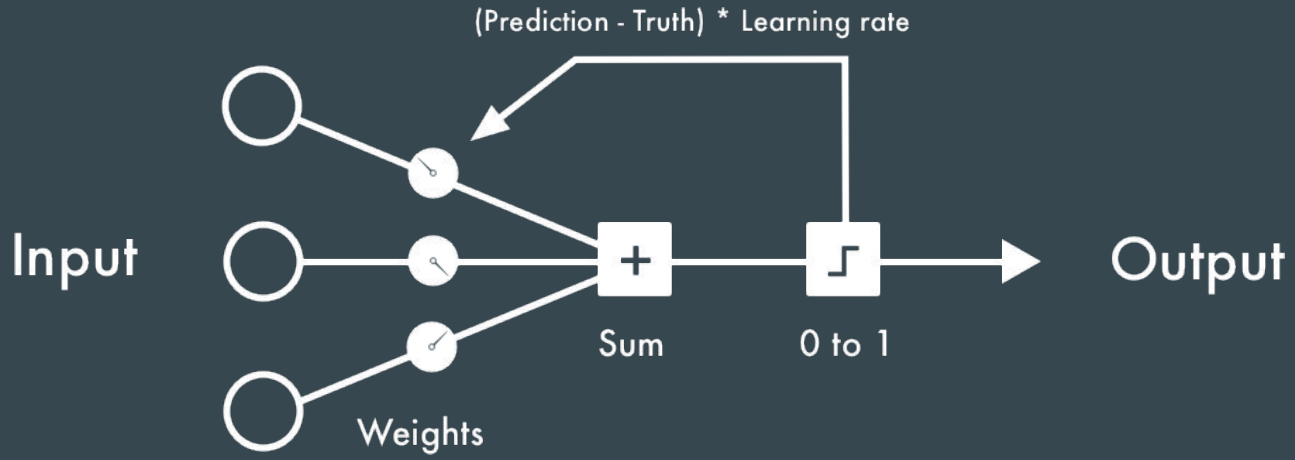




Deep Learning. Perceptron.

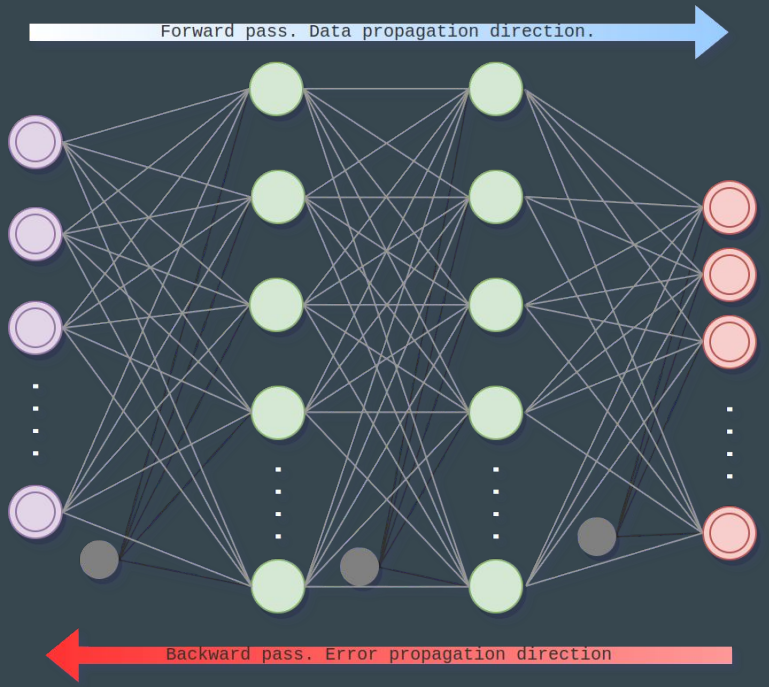
Perceptron - Frank Rosenblatt (1958)

- Use layers of neurons as computation tool
- Proposes a training algorithm



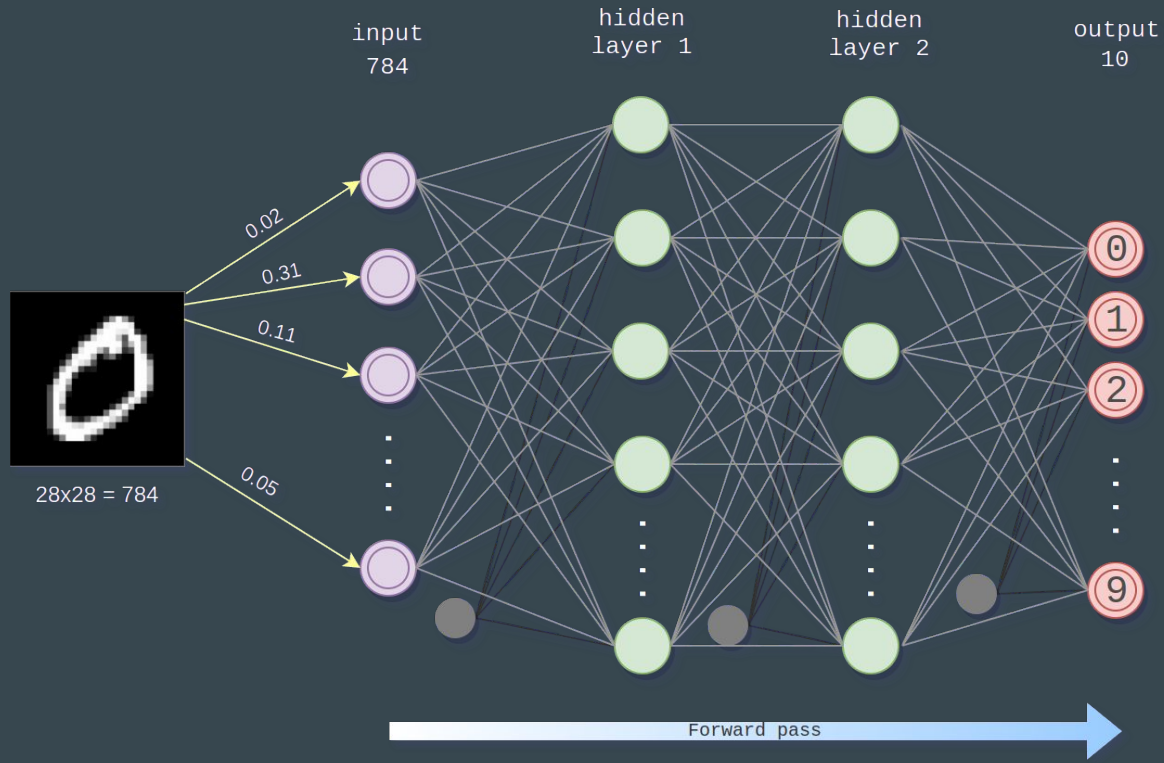
Deep Learning. Backpropagation.

Paul Werbos introduced Backpropagation (1974)



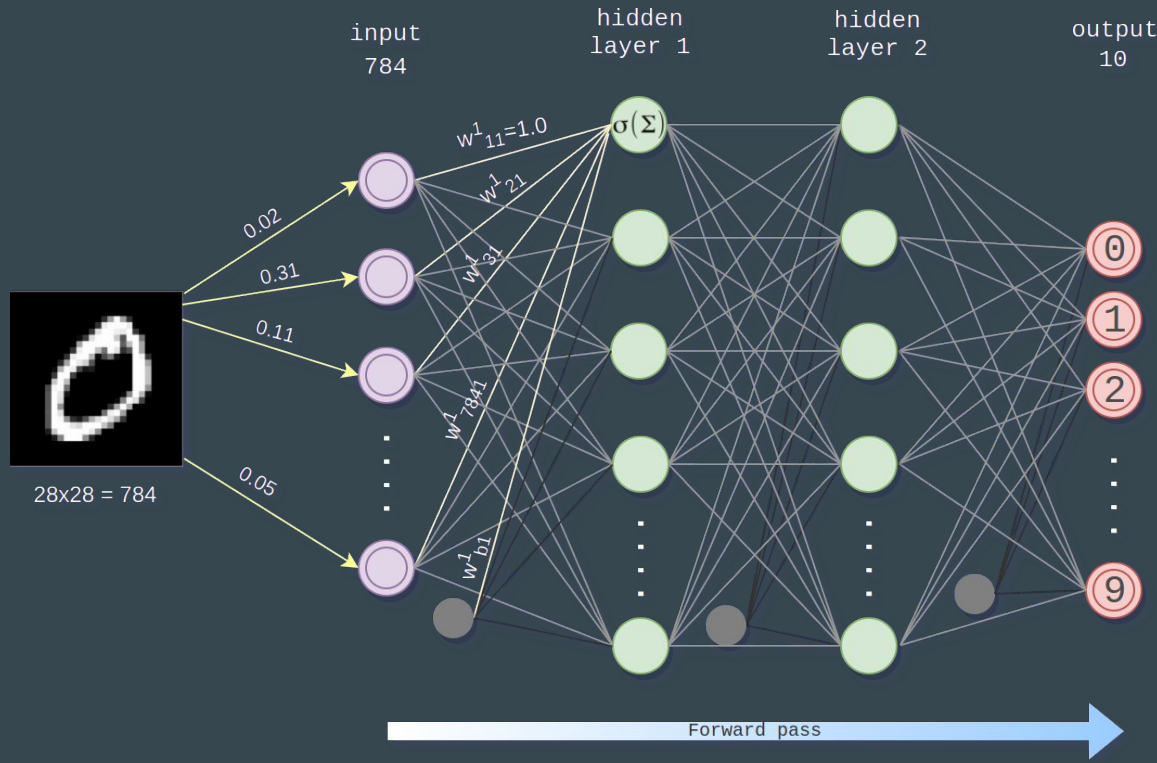
1. Propagate data from input to output.
2. Calculate the gradient of the loss function w.r.t the weights.
3. Update weights to minimize error.

Forward Pass.



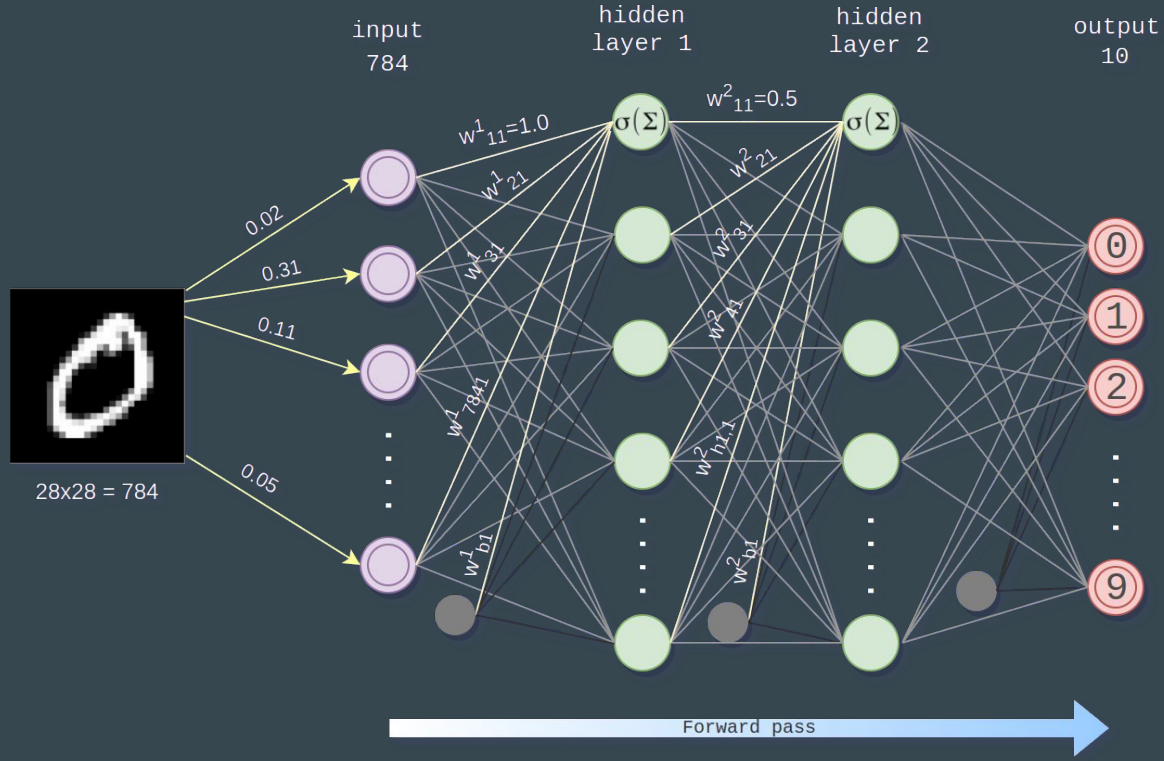
$$\mathrm{a}_n^L = \left[\sigma\left(\sum_m w_{nm}^L \left[\cdots\left[\sigma\left(\sum_j w_{kj}^2\left[\sigma\left(\sum_i w_{ji}^1 x_i + b_j^1\right)\right] + b_k^2\right)\right]\cdots\right]_m + b_n^L\right)\right]_n$$

Forward Pass.



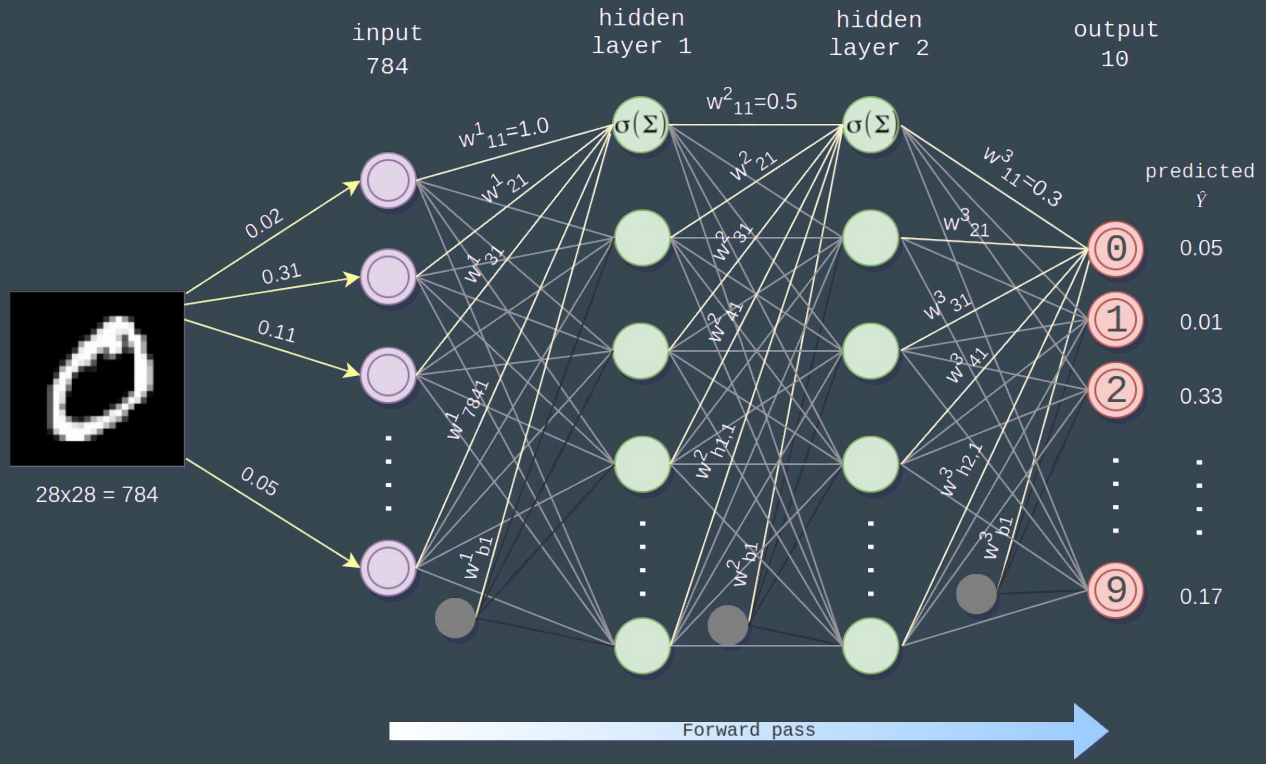
$$a_n^L = \left[\sigma\left(\sum_m w_{nm}^L \left[\cdots\left[\sigma\left(\sum_j w_{kj}^2\left[\sigma\left(\sum_i w_{ji}^1 x_i + b_j^1\right)\right] + b_k^2\right)\right]\cdots\right]_m + b_n^L\right)\right]_n$$

Forward Pass.



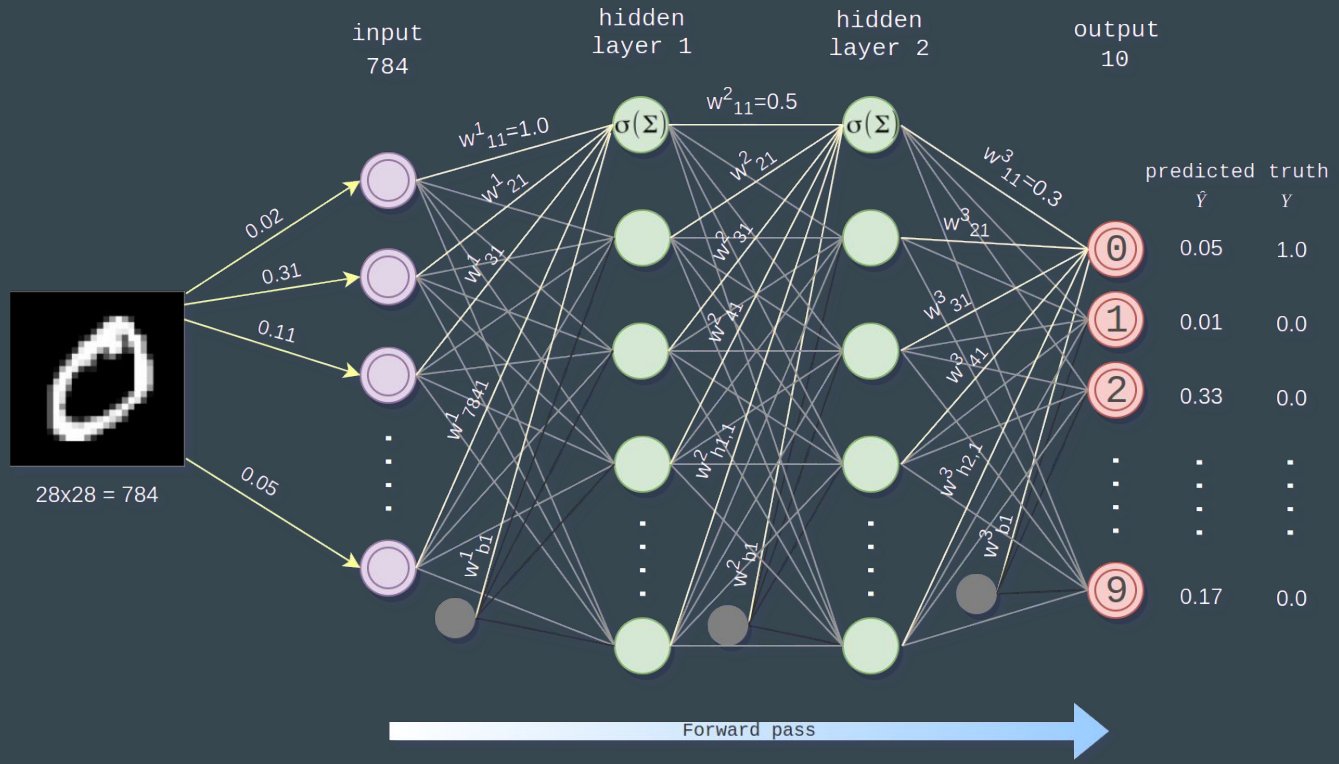
$$a_n^L = \left[\sigma\left(\sum_m w_{nm}^L \left[\cdots\left[\sigma\left(\sum_j w_{kj}^2\left[\sigma\left(\sum_i w_{ji}^1 x_i + b_j^1\right)\right] + b_k^2\right)\right]\cdots\right]_m + b_n^L\right)\right]_n$$

Forward Pass.



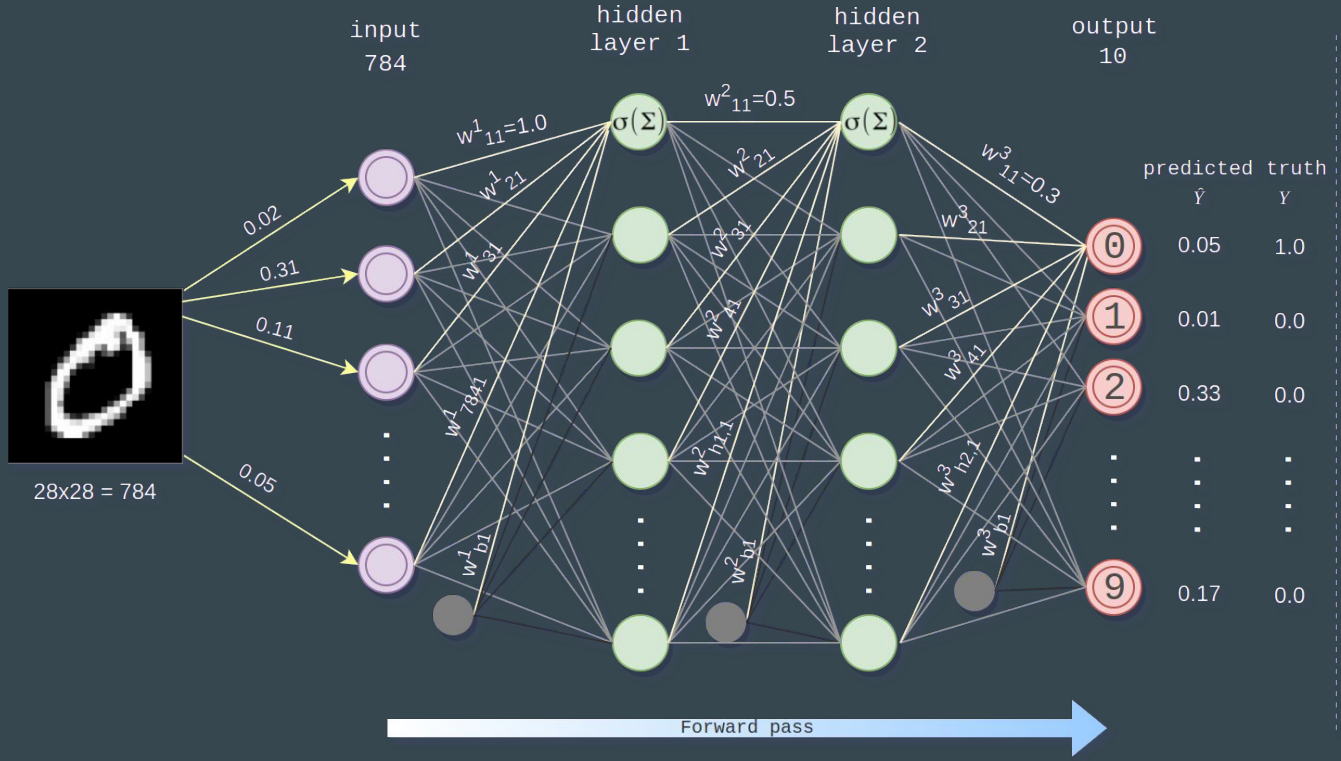
$$a_n^L = \left[\sigma\left(\sum_m w_{nm}^L \left[\cdots\left[\sigma\left(\sum_j w_{kj}^2\left[\sigma\left(\sum_i w_{ji}^1 x_i + b_j^1\right)\right] + b_k^2\right)\right]\cdots\right]_m + b_n^L\right)\right]_n$$

Forward Pass.



$$a_n^L = \left[\sigma\left(\sum_m w_{nm}^L \left[\cdots\left[\sigma\left(\sum_j w_{kj}^2\left[\sigma\left(\sum_i w_{ji}^1 x_i + b_j^1\right)\right] + b_k^2\right)\right]\cdots\right]_m + b_n^L\right)\right]_n$$

Forward Pass.



cost -> MSE

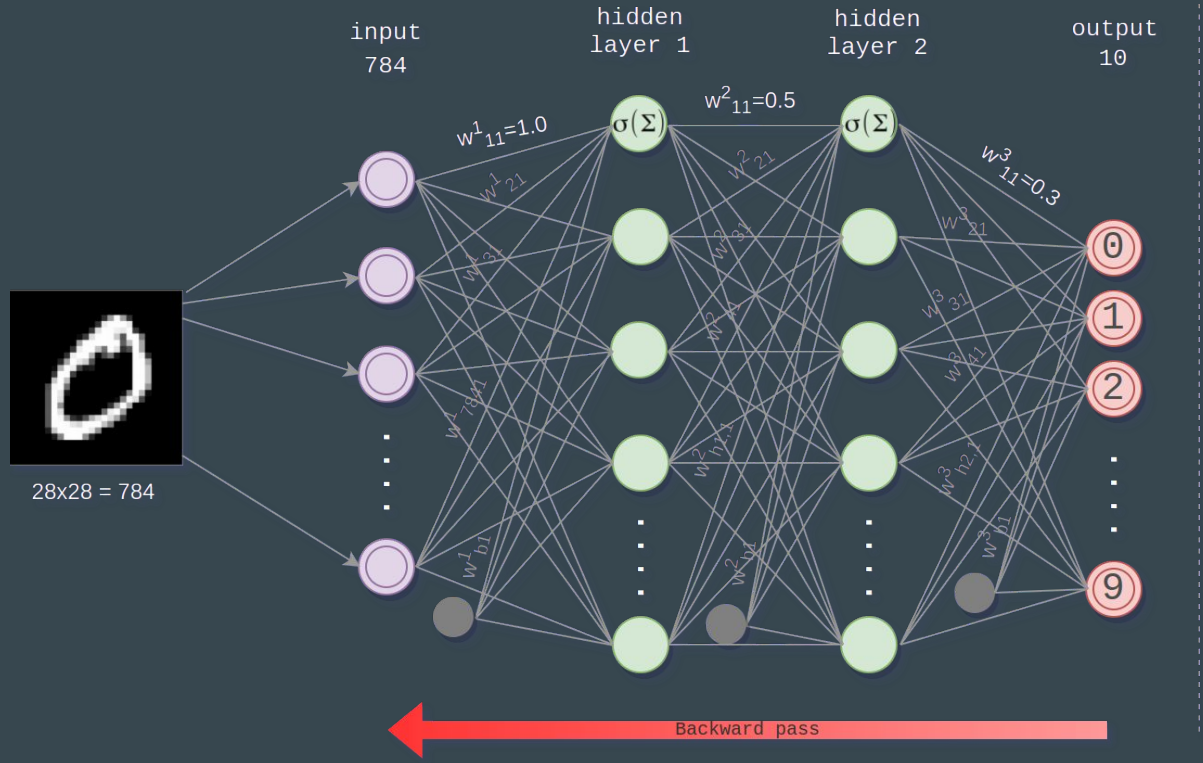
Mean Squared Error

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$$

$$MSE = \frac{0.95^2+0.01^2+0.33^2+...0.17^2}{10}$$

$$\mathrm{a}^L_n = \left[\sigma\left(\sum_m w^L_{nm}\left[\cdots\left[\sigma\left(\sum_j w^2_{kj}\left[\sigma\left(\sum_i w^1_{ji}x_i + b^1_j\right)\right] + b^2_k\right)\right]\cdots\right]_m + b^L_n\right)\right]_n$$

Backward Pass.



cost -> MSE

Mean Squared Error

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$$

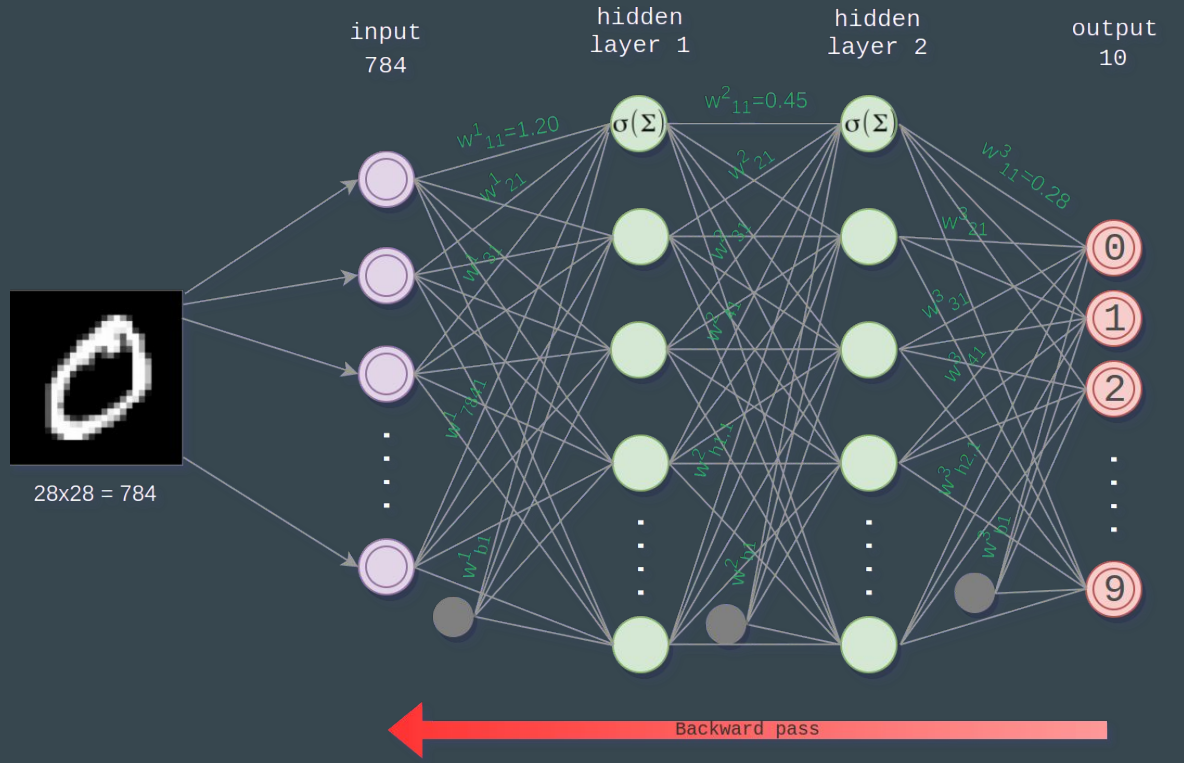
Compute gradients w.r.t. weights

$$\frac{\partial MSE}{\partial w^1_{11}} = -2.0 \qquad \frac{\partial MSE}{\partial w^2_{11}} = 0.5 \qquad \frac{\partial MSE}{\partial w^3_{11}} = 0.2$$

$$\frac{\partial MSE}{\partial w^1_{21}} = -1.1 \qquad \frac{\partial MSE}{\partial w^2_{21}} = 0.5 \qquad \frac{\partial MSE}{\partial w^3_{21}} = -0.3$$

$$W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{(l)}} J(W,b) \qquad b_i^{(l)} = b_i^{(l)} - \alpha \frac{\partial}{\partial b_i^{(l)}} J(W,b)$$

Backward Pass.



$$W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{(l)}} J(W,b) \qquad b_i^{(l)} = b_i^{(l)} - \alpha \frac{\partial}{\partial b_i^{(l)}} J(W,b)$$

cost -> MSE

Mean Squared Error

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$$

↓

Compute gradients w.r.t. weights

$$\frac{\partial MSE}{\partial w^1_{11}} = -2.0 \qquad \frac{\partial MSE}{\partial w^2_{11}} = 0.5 \qquad \frac{\partial MSE}{\partial w^3_{11}} = 0.2$$

$$\frac{\partial MSE}{\partial w^1_{21}} = -1.1 \qquad \frac{\partial MSE}{\partial w^2_{21}} = 0.5 \qquad \frac{\partial MSE}{\partial w^3_{21}} = -0.3$$

⋮ ↓ ⋮ ⋮

Upgrade weights

$$w^l_{ij} \leftarrow w^l_{ij} - \delta \cdot \frac{\partial MSE}{\partial w^l_{ij}}$$

δ is the *learning rate*

with $\delta = 0.1$:

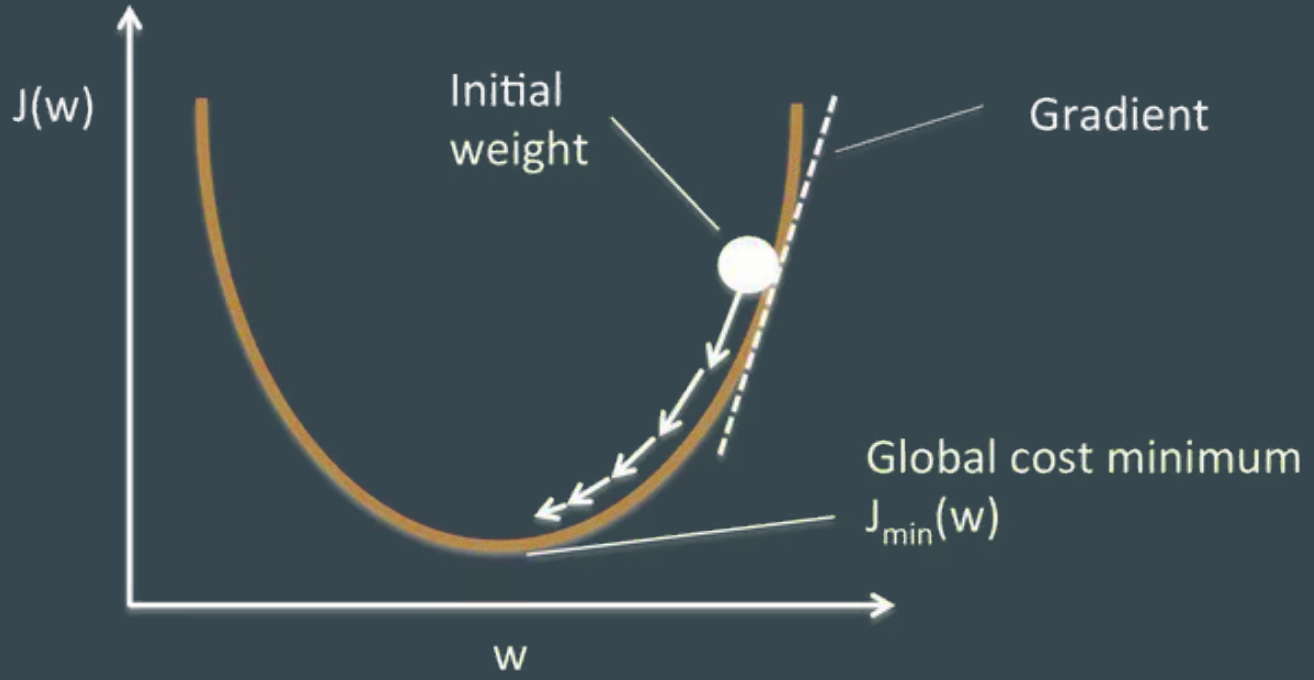
$$w^3_{11} \leftarrow w^3_{11} - \delta \cdot \frac{\partial MSE}{\partial w^3_{11}} = 0.3 - 0.1 \cdot 0.2 = 0.28$$

$$w^2_{11} \leftarrow w^2_{11} - \delta \cdot \frac{\partial MSE}{\partial w^2_{11}} = 0.5 - 0.1 \cdot 0.5 = 0.45$$

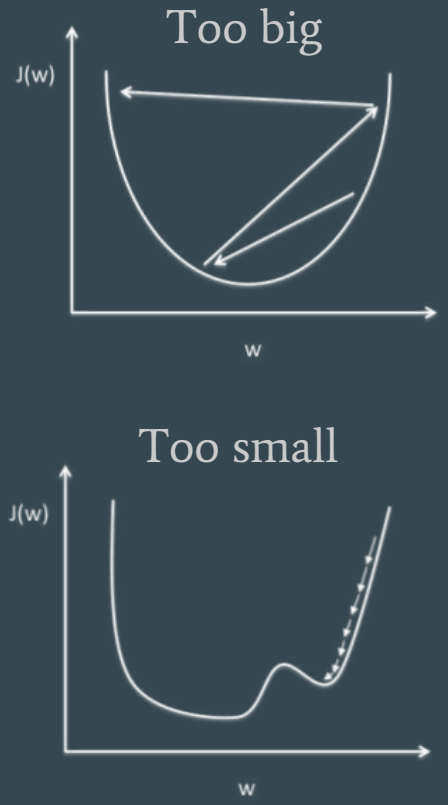
$$w^1_{11} \leftarrow w^1_{11} - \delta \cdot \frac{\partial MSE}{\partial w^1_{11}} = 1.0 - 0.1 \cdot (-2.0) = 1.20$$

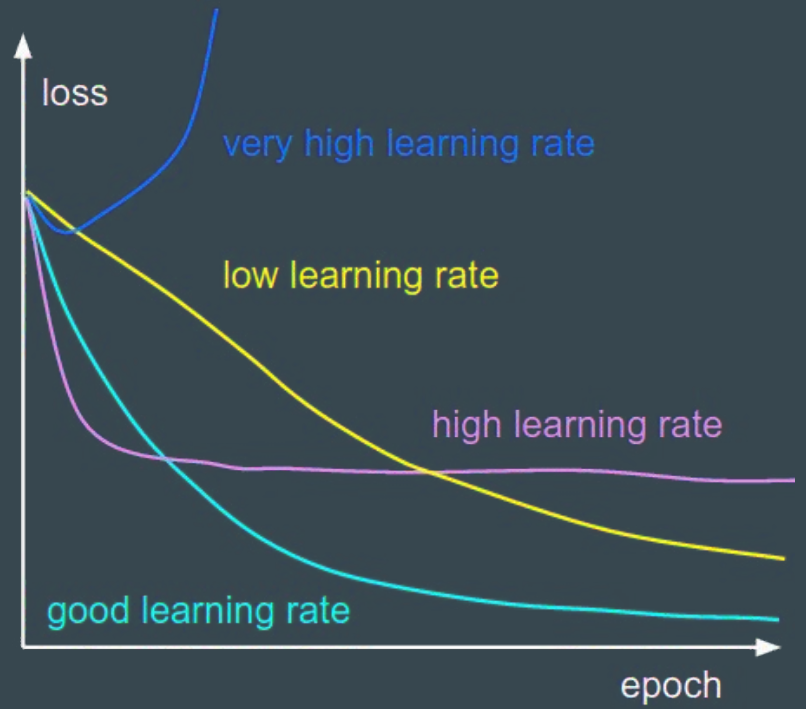
⋮

Deep Learning. Learning rate.

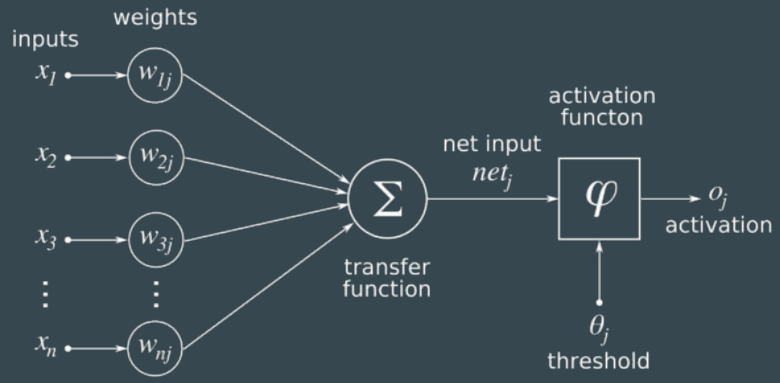


Deep Learning. Learning rate.

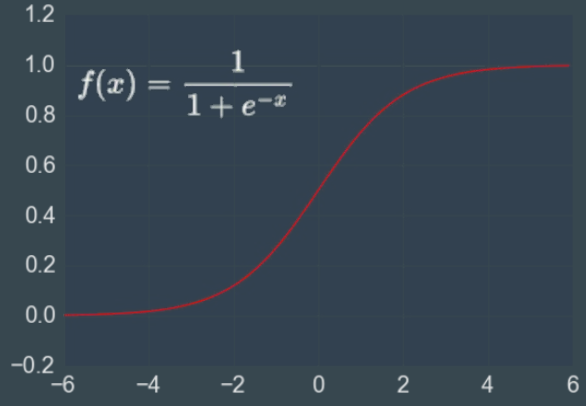




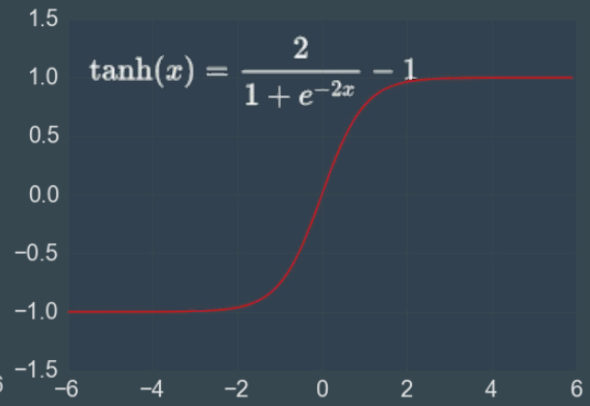
Deep Learning. Activation functions.



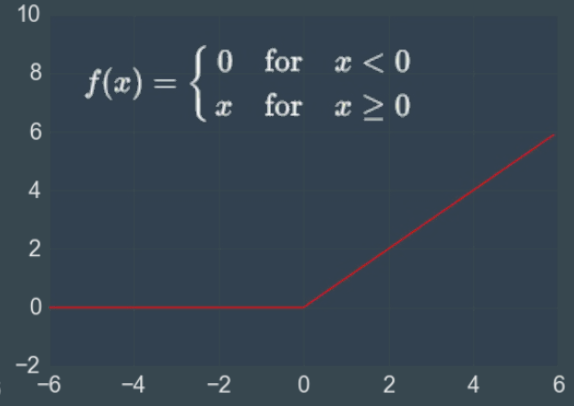
Sigmoid



TanH



ReLU



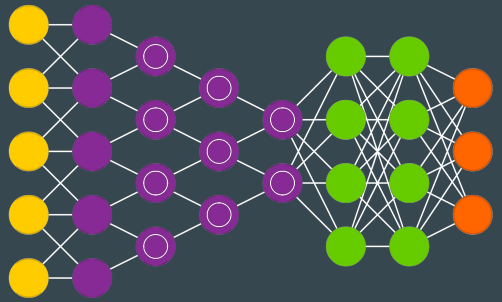
Deep Learning. Neural Network Types.

Perceptron (P)

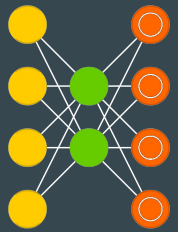
Feed Forward (FF)

Deep Feed Forward (DFF)

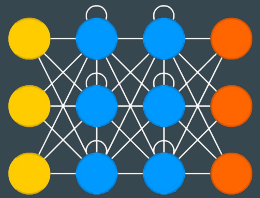
Deep Convolutional Network (DCN)



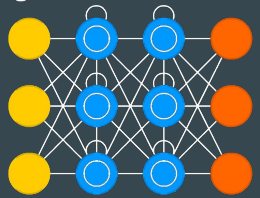
Auto Encoder (AE)



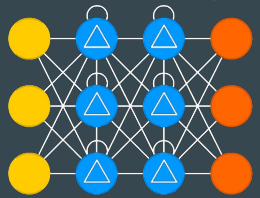
Recurrent Neural Network (RNN)



Long / Short Term Memory (LSTM)



Gated Recurrent Unit (GRU)



Implementing Deep Learning algorithms.

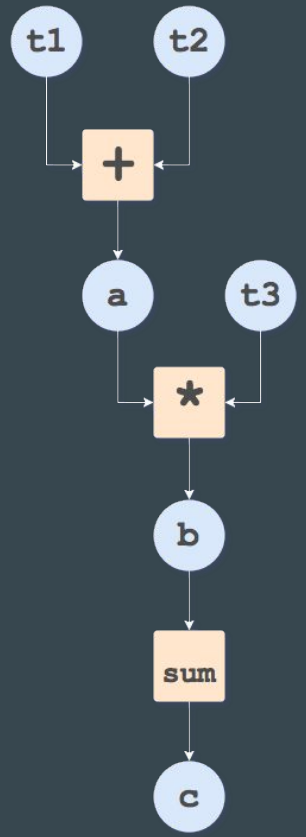
Deep Learning Software.

- Caffe (UC Berkeley) / Caffe2 (Facebook)
- Theano (U Montreal) / Tensorflow (Google)
- Torch (NYU / Facebook) / PyTorch (Facebook)
- Paddle (Baidu), CNTK (Microsoft), MXNet (Amazon) ...

Computational Graph with Numpy.

```python
import numpy as np
np.random.seed(0)

# Tensor dimensions
d1, d2 = 4, 5

# Initialize three tensors
t1 = np.random.rand(d1, d2)
t2 = np.random.rand(d1, d2)
t3 = np.random.rand(d1, d2)

# Computational graph
a = t1 + t2
b = a * t3
c = np.sum(b) # 8.80812307798
```

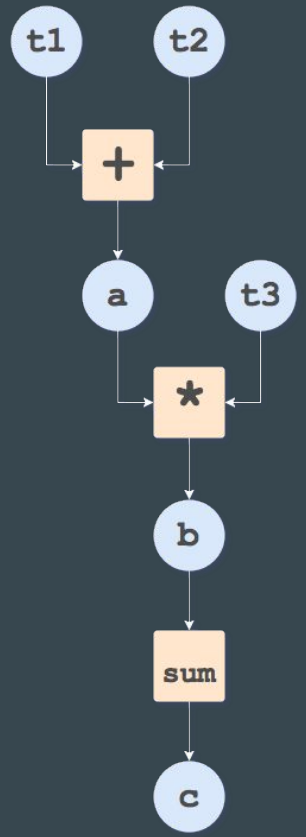
```python
# Gradient computation
grad_c = 1.0
grad_b = grad_c * np.ones((d1, d2))
grad_a = grad_b * t3
grad_t3 = grad_b * a
grad_t1 = grad_a.copy()
grad_t2 = grad_a.copy()
```



Computational Graph with Tensorflow

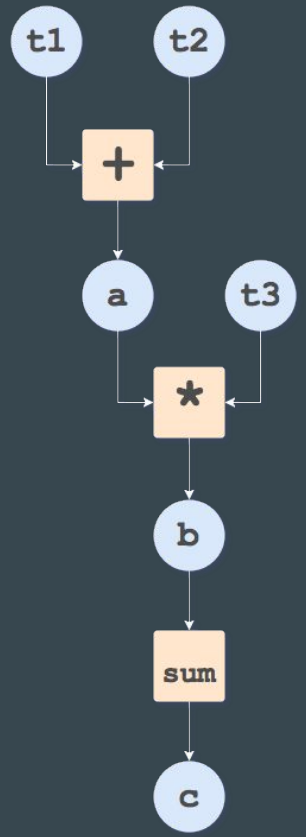
```
import numpy as np
import tensorflow as tf
np.random.seed(0)

# Tensor dimensions
d1, d2 = 4, 5

# Computational graph definition
t1 = tf.placeholder(tf.float32)
t2 = tf.placeholder(tf.float32)
t3 = tf.placeholder(tf.float32)
a = t1 + t2
b = a * t3
c = tf.reduce_sum(b)

# Gradients definition
grad_t1, grad_t2, grad_t3 =
tf.gradients(c, [t1, t2, t3])
```

```
# Computation
with tf.Session() as sess:
    values = {
        t1: np.random.rand(d1, d2),
        t2: np.random.rand(d1, d2),
        t3: np.random.rand(d1, d2),
    }

    out = sess.run([c, grad_t1,
      grad_t2, grad_t3],
      feed_dict=values)

    c_val, grad_t1_val,
      grad_t2_val, grad_t3_val = out
```



Computational Graph with PyTorch

```
import torch
from torch.autograd import Variable
torch.manual_seed(0)

# Tensor dimensions
d1, d2 = 4, 5

# Computational graph definition with Variables
t1 = Variable(torch.randn(d1, d2), requires_grad=True)
t2 = Variable(torch.randn(d1, d2), requires_grad=True)
t3 = Variable(torch.randn(d1, d2), requires_grad=True)
a = t1 + t2
b = a * t3
c = torch.sum(b)

# Gradient computation
c.backward()

grad_t1, grad_t2, grad_t3 = t1.grad.data, t2.grad.data, t3.grad.data
```



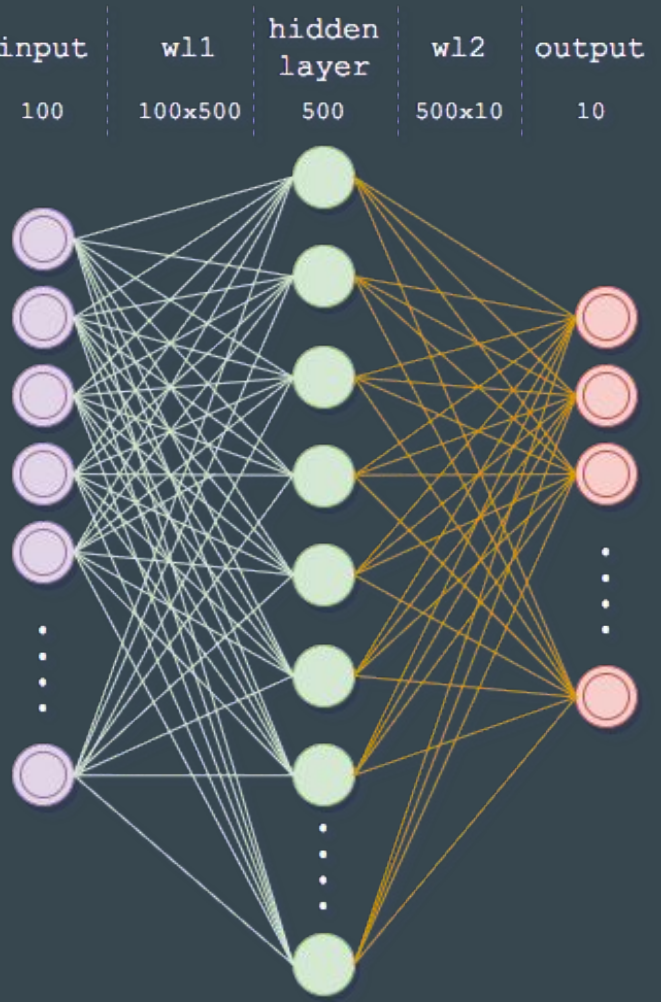
Deep Learning with PyTorch

Abstraction levels:

1. Tensor: Array that can run on GPU
2. Variable: Node in a computational graph. Stores data and gradient.
3. Module: A NN layer. May store state or learnable weights. Higher level wrapper for working with neural nets.

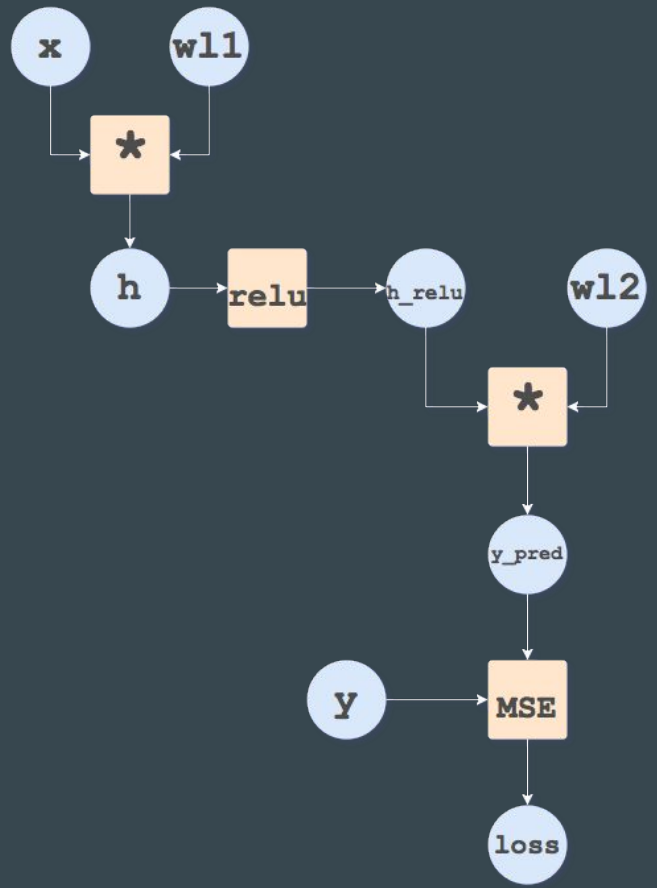
PyTorch: Tensors [1]

```
import torch
torch.manual_seed(0)

# Neural network dimensions
n, in_dim, hid_dim, out_dim = 50, 100, 500, 10

# Training data
x = torch.randn(n, in_dim).type(torch.FloatTensor)
y = torch.randn(n, out_dim).type(torch.FloatTensor)

# Weight inizialization
wl1 = torch.randn(in_dim, hid_dim).type(torch.FloatTensor)
wl2 = torch.randn(hid_dim, out_dim).type(torch.FloatTensor)
```



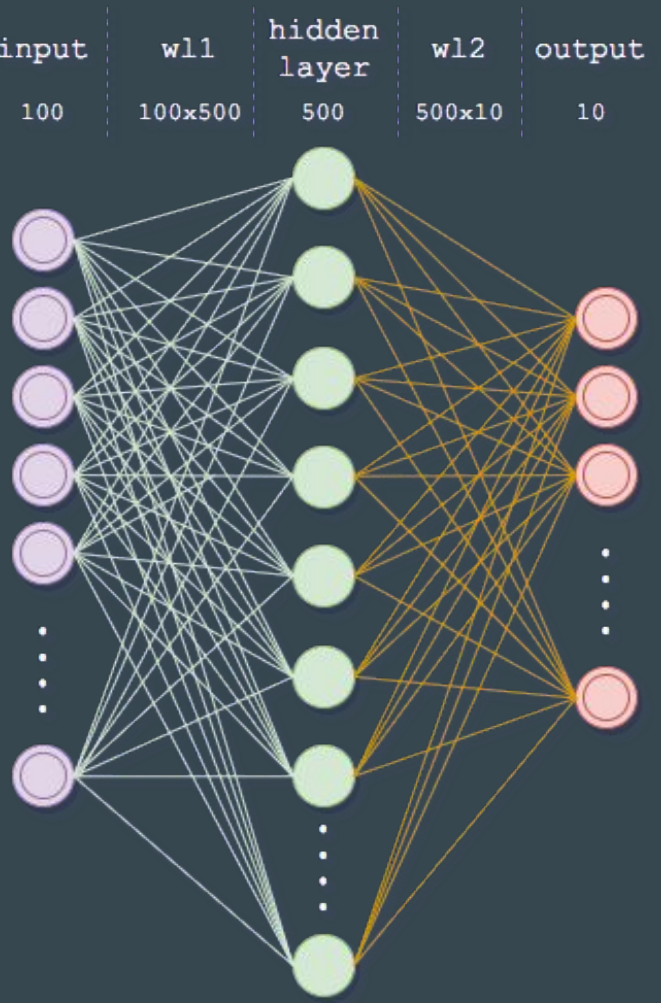
PyTorch: Tensors [2]

```
learning_rate = 1e-6
for epoch in range(500):

    # Computational graph
    h = x.mm(w1)
    h_relu = h.clamp(min=0)
    y_pred = h_relu.mm(w2)
    loss = (y_pred - y).pow(2).sum()

    # Gradient computation
    grad_y_pred = 2.0 * (y_pred - y)
    grad_w2 = h_relu.t().mm(grad_y_pred)
    grad_h_relu = grad_y_pred.mm(w2.t())
    grad_h = grad_h_relu.clone()
    grad_h[grad_h < 0] = 0
    grad_w1 = x.t().mm(grad_h)

    # Update weights
    w1 -= learning_rate * grad_w1
    w2 -= learning_rate * grad_w2
```



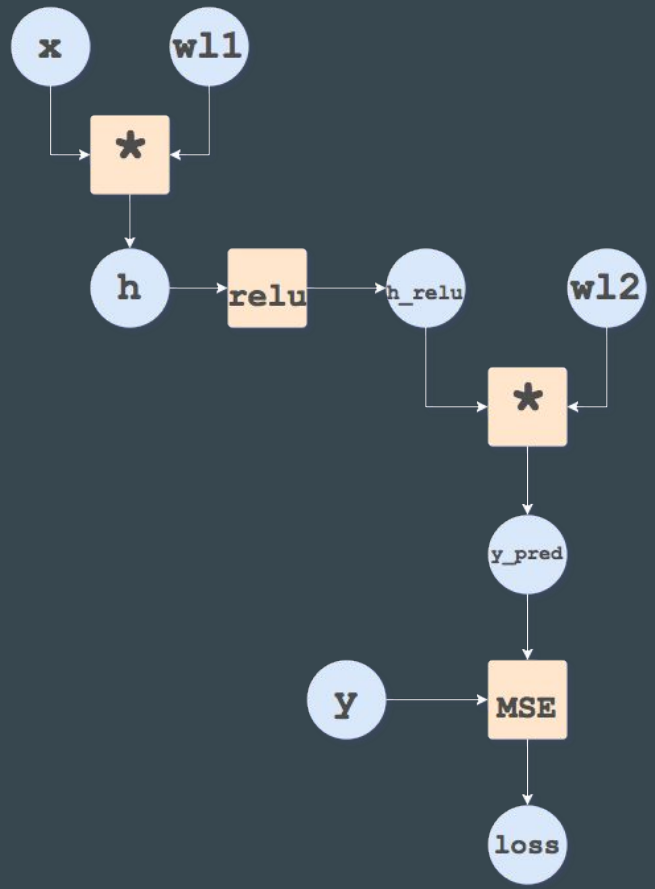
PyTorch: Variable & Autograd [1]

```
import torch
from torch.autograd import Variable
torch.manual_seed(0)

# Neural network dimensions
n, in_dim, hid_dim, out_dim = 50, 100, 500, 10

# Training data
x = Variable(torch.randn(n, in_dim), requires_grad=False)
y = Variable(torch.randn(n, out_dim), requires_grad=False)

# Weight inizialization
wl1 = Variable(torch.randn(in_dim, hid_dim),
               requires_grad=True)
wl2 = Variable(torch.randn(hid_dim, out_dim),
               requires_grad=True)
```



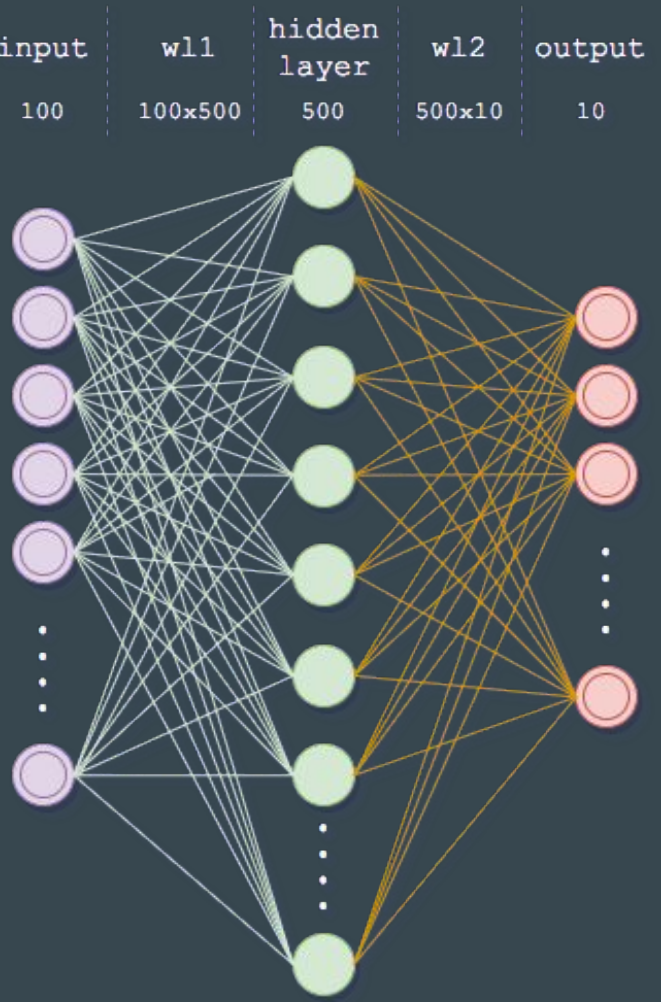
PyTorch: Variable & Autograd [2]

```
learning_rate = 1e-6
for epoch in range(500):

    # Computational graph
    h = x.mm(w1)
    h_relu = h.clamp(min=0)
    y_pred = h_relu.mm(w2)
    loss = (y_pred - y).pow(2).sum()

    # Remove previous gradients
    if w1.grad is not None: w1.grad.data.zero_()
    if w2.grad is not None: w2.grad.data.zero_()

    # Gradient computation
    loss.backward()

    # Update weights
    w1.data -= learning_rate * w1.grad.data
    w2.data -= learning_rate * w2.grad.data
```



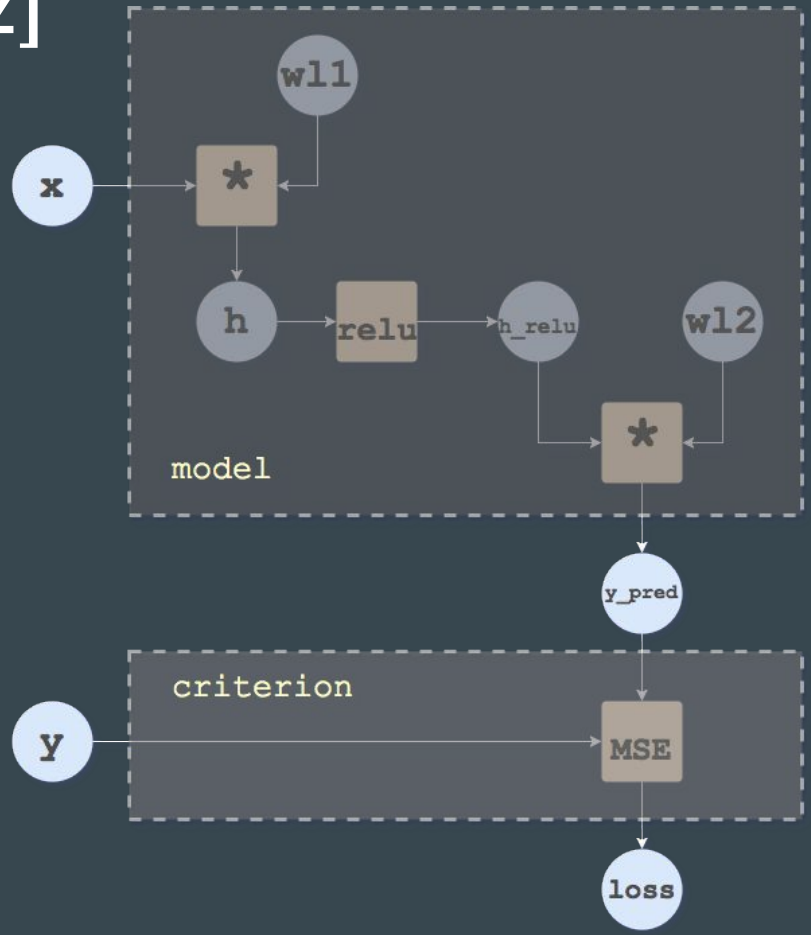
PyTorch modules: nn & optim [1]

```
import torch
from torch.autograd import Variable
torch.manual_seed(0)

# Neural network dimensions
n, in_dim, hid_dim, out_dim = 50, 100, 500, 10

# Training data
x = Variable(torch.randn(n, in_dim), requires_grad=False)
y = Variable(torch.randn(n, out_dim), requires_grad=False)

# Model definition
model = torch.nn.Sequential(
   torch.nn.Linear(in_dim, hid_dim),
   torch.nn.ReLU(),
   torch.nn.Linear(hid_dim, out_dim)
)
```

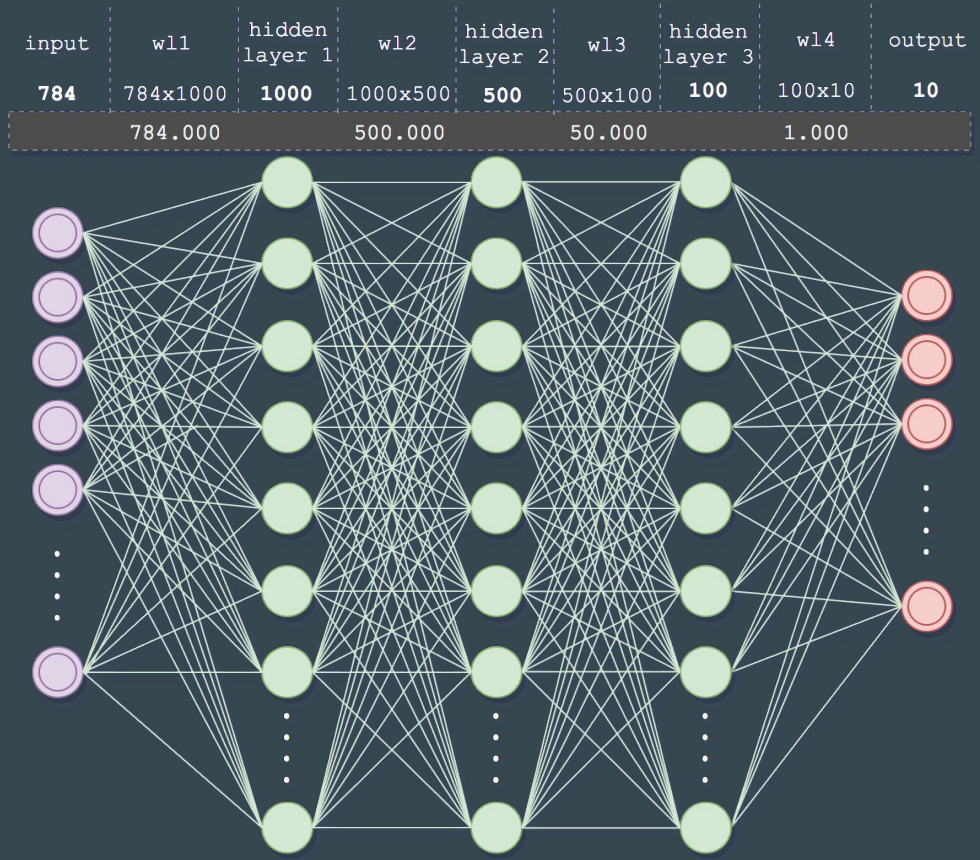


PyTorch modules: nn & optim [2]

```
# Loss function
criterion = torch.nn.MSELoss()

# Optimization function
learning_rate = 1e-6
optimizer = torch.optim.SGD(model.parameters(),
                            learning_rate)

for epoch in range(500):

    # Forward pass
    y_pred = model(x)
    loss = criterion(y_pred, y)

    # Backward pass
    optimizer.zero_grad()
    loss.backward()

    # Update weights
    optimizer.step()
```



PyTorch example. MNIST classifier (1/3)



PyTorch example. MNIST classifier (2/3)

```
import torch
import pandas as pd
import numpy as np
from torch.autograd import Variable
from sklearn.model_selection import
train_test_split
from sklearn.preprocessing import OneHotEncoder

torch.manual_seed(0)

# Load data
train = pd.read_csv('train.csv', header=0).values
x = train[:, 1:] / 255.0
y = train[:, 0]
enc = OneHotEncoder()
y = enc.fit_transform(y.reshape(-1, 1)).toarray()
x_train, x_test, y_train, y_test =
train_test_split(x, y, test_size=0.33,
                 random_state=0)
```

```
# Neural network dimensions
in_dim = x_train.shape[1]
hid_dim_1 = 1000
hid_dim_2 = 500
hid_dim_3 = 100
out_dim = 10

# Batch size
batch = 1000

# Training data Variables
x = Variable(torch.from_numpy(x_train),
requires_grad=False).type(torch.FloatTensor)

y = Variable(torch.from_numpy(y_train),
requires_grad=False).type(torch.FloatTensor)

# Test data Variables
xt = Variable(torch.from_numpy(x_test),
requires_grad=False).type(torch.FloatTensor)
```

PyTorch example. MNIST classifier (3/3)

```
# Model definition
model = torch.nn.Sequential(
    torch.nn.Linear(in_dim, hid_dim_1),
    torch.nn.ReLU(),
    torch.nn.Linear(hid_dim_1, hid_dim_2),
    torch.nn.ReLU(),
    torch.nn.Linear(hid_dim_2, hid_dim_3),
    torch.nn.ReLU(),
    torch.nn.Linear(hid_dim_3, out_dim)
)

# Loss function
criterion = torch.nn.MSELoss()

# Learning rate
learning_rate = 1

# Optimization function
optimizer = torch.optim.SGD(model.parameters(),
                            learning_rate)
```

```
for epoch in range(100):
    epoch_loss = 0
    for i in range(0, x.size()[0], batch):
        # Forward pass
        y_pred = model(x[i:(i+batch), :])
        loss = criterion(y_pred, y[i:(i+batch), :])
        epoch_loss += loss.data[0]

        # Backward pass
        optimizer.zero_grad()
        loss.backward()

        # Update weights
        optimizer.step()

    # Test accuracy
    yt_pred = model(xt).data.numpy()
    accuracy = np.sum(np.argmax(y_test, axis=1) ==
                      np.argmax(yt_pred, axis=1)) /
                      yt_pred.shape[0]

    print('Epoch: %d, Loss: %f, Accuracy: %f' %
          (epoch + 1, epoch_loss, accuracy))
```

https://github.com/esansano/decharlas_deep_learning


FIN