Training Neural Networks

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g\left(z^{(2)}\right)$$

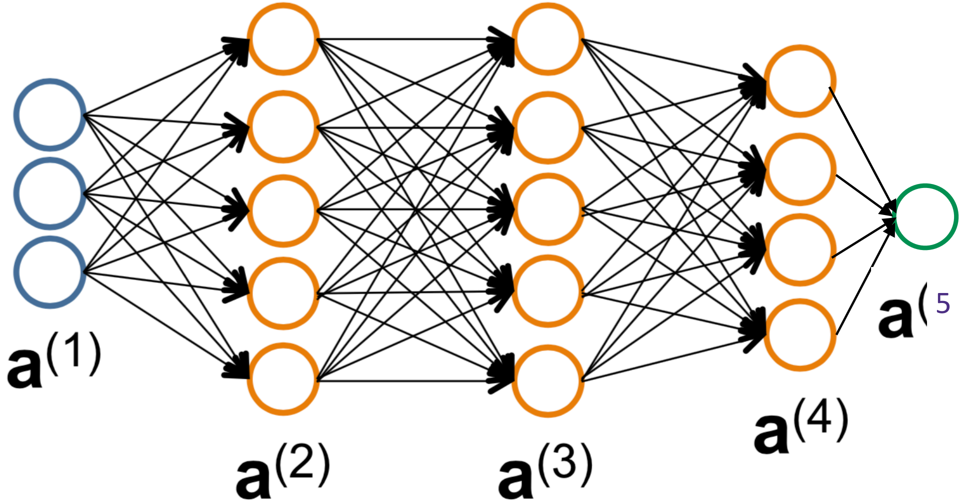
$$\vdots$$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g\left(z^{(l+1)}\right)$$

$$\vdots$$

$$\hat{y} = g(\Theta^{(L)} a^{(L)})$$



$$L(y, \hat{y}) = y \log(\hat{y}) + (1 - y)\log(1 - \hat{y})$$

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

Gradient Descent: $\Theta^{(l)} \leftarrow \Theta^{(l)} - \eta \nabla_{\Theta^{(l)}} L(y, \hat{y}) \qquad \forall l$

Gradient Descent: $\Theta^{(l)} \leftarrow \Theta^{(l)} - \eta \nabla_{\Theta^{(l)}} L(y, \widehat{y}) \qquad \forall l$

Seems simple enough, why are packages like PyTorch, Tensorflow, Theano, Cafe, MxNet synonymous with deep learning?

1. Automatic differentiation

① compute gradient automatically & efficiently

2. Convenient libraries

② set up NN
③ training
③ tune hyper-parameters

3. GPU support

① linear algebra operations
② pointwise operation

Gradient Descent:

Seems simple enough, Theano, Cafe, MxNet s

1. Automatic differ

2. Convenient libra

```
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        # 1 input image channel, 6 output channels, 3x3 square convolution
        # kernel
        self.conv1 = nn.Conv2d(1, 6, 3)
        self.conv2 = nn.Conv2d(6, 16, 3)
        # an affine operation: y = Wx + b
        self.fc1 = nn.Linear(16 * 6 * 6, 120)  # 6*6 from image dimension
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        # Max pooling over a (2, 2) window
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        # If the size is a square you can only specify a single number
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(-1, self.num_flat_features(x))
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

```
# create your optimizer
optimizer = optim.SGD(net.parameters(), lr=0.01)

# in your training loop:
optimizer.zero_grad()   # zero the gradient buffers
output = net(input)
loss = criterion(output, target)
loss.backward()
optimizer.step()    # Does the update
```

Recommendation: Start with existing implementation (hyper-parameter)

Common training issues

Neural networks are **non-convex**

- For large networks, **gradients** can **blow up** or **go to zero**. This can be helped by **batchnorm** or ResNet architecture

architecture tricks

- **Stepsize**, **batchsize**, **momentum** all have large impact on optimizing the training error *and* generalization performance

- Fancier alternatives to SGD (Adagrad, Adam, LAMB, etc.) can significantly improve training

- Overfitting is common and not undesirable: typical to achieve 100% training accuracy even if test accuracy is just 80%

- Making the network *bigger* may make training *faster!*

over-parameterization

Common training issues

Training is too slow:

divide step size by 10 every 100 iterations

- Use larger step sizes, develop step size reduction schedule
- Use GPU resources
- Change batch size
- Use momentum and more exotic optimizers (e.g., Adam)
- Apply batch normalization
- Make network larger or smaller (# layers, # filters per layer, etc.)

Test accuracy is low

- Try modifying all of the above, plus changing other hyperparameters

Intuition

https://playground.tensorflow.org/

Back Propagation

Forward Propagation

(ignore bias)

$z^{(l)}$: pre-activation
g: activation function

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g\left(z^{(2)}\right)$$

$$\vdots$$

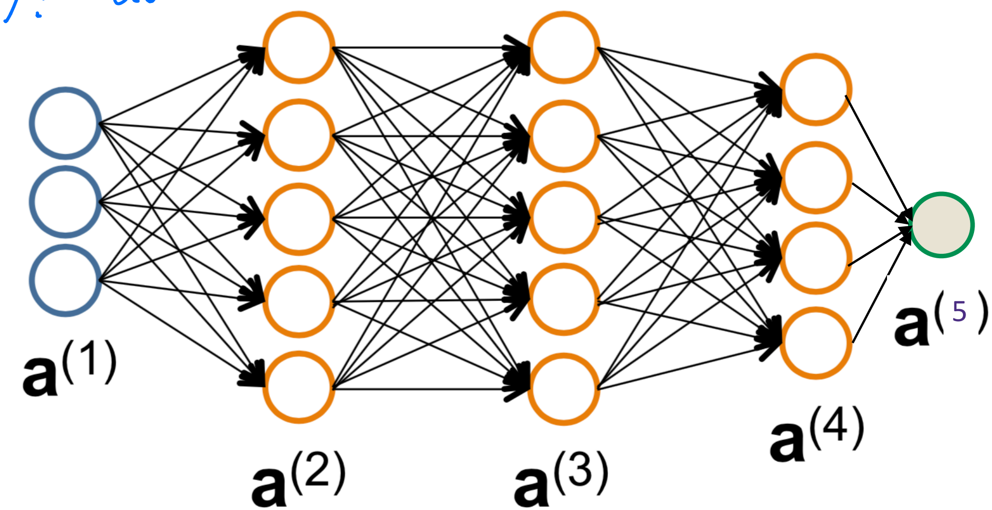
$$a^{(l)} = g(z^{(l)})$$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g\left(z^{(l+1)}\right)$$

$$\vdots$$

$$\hat{y} = a^{(L+1)}$$



$$L(y, \hat{y}) = y \log(\hat{y}) + (1-y)\log(1-\hat{y})$$

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

Backprop

$\Theta^{(1)}, \Theta^{(2)}, \dots \Theta^{(L)}$

$\Theta^{(1)} \in \mathbb{R}^{m \times d}, \Theta^{(2)}, \dots, \Theta^{(L-1)} \in \mathbb{R}^{m\times m}, \Theta^{(L)} \in \mathbb{R}^{m}$

$$a^{(1)} = x \quad \in \mathbb{R}^d$$

$$z^{(2)} = \Theta^{(1)} a^{(1)} \quad \in \mathbb{R}^m$$

$$a^{(2)} = g\left(z^{(2)}\right) \quad \in \mathbb{R}^m$$

$$\vdots$$

$$a^{(l)} = g(z^{(l)}) \quad \in \mathbb{R}^m$$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)} \quad \in \mathbb{R}^m$$

$$a^{(l+1)} = g\left(z^{(l+1)}\right)$$

$$z^{(L+1)} = \Theta^{(L)} a^{(L)} \quad \in \mathbb{R}$$

$$\hat{y} = a^{(L+1)} = g(z^{(L+1)}) \quad \in \mathbb{R}$$

Train by Stochastic Gradient Descent:

$\Theta_{i,j}^{(l)} \in \mathbb{R}^{m\times m}$

$i = 1, \dots, m$
$j = 1, \dots, m$

$$\Theta_{i,j}^{(l)} \leftarrow \Theta_{i,j}^{(l)} - \eta \frac{\partial L(y, \hat{y})}{\partial \Theta_{i,j}^{(l)}}$$

$$L(y, \hat{y}) = y \log(\hat{y}) + (1 - y)\log(1 - \hat{y})$$

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

intermediate quantity

$$\delta_i^{(l+1)} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}}$$

if $l = L$ $\in \mathbb{R}$
o.w. $\in \mathbb{R}^m$

Backprop

chain Rule

for $1 \leq L$

$\delta_i \in \mathbb{R}$, $a_j^{(l)} \in \mathbb{R}$

$$\frac{\partial L(y,\hat{y})}{\partial \Theta_{i,j}^{(l)}} = \frac{\partial L(y,\hat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta_{i,j}^{(l)}} =: \delta_i^{(l+1)} \cdot \underbrace{a_j^{(l)}}_{\text{computed in the forward pass}}$$

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g\left(z^{(2)}\right)$$

$$\vdots$$

$$a^{(l)} = g(z^{(l)})$$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g\left(z^{(l+1)}\right)$$

$$\vdots$$

$$\hat{y} = a^{(L+1)}$$

Train by Stochastic Gradient Descent:

$$\Theta_{i,j}^{(l)} \leftarrow \Theta_{i,j}^{(l)} - \eta \frac{\partial L(y,\hat{y})}{\partial \Theta_{i,j}^{(l)}}$$

$\Theta_{i,j}^{(l)}$: link from $a_j^{(l)}$ to $z_i^{(l+1)}$

$$L(y,\hat{y}) = y\log(\hat{y}) + (1-y)\log(1-\hat{y})$$

$$g(z) = \frac{1}{1+e^{-z}} \qquad \delta_i^{(l+1)} = \frac{\partial L(y,\hat{y})}{\partial z_i^{(l+1)}}$$

Backprop

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g\left(z^{(2)}\right)$$

$$\vdots$$

$$a^{(l)} = g(z^{(l)})$$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g\left(z^{(l+1)}\right)$$

$$\vdots$$

$$\hat{y} = a^{(L+1)}$$

$$\frac{\partial L(y, \hat{y})}{\partial \Theta^{(l)}_{i,j}} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta^{(l)}_{i,j}} =: \delta_i^{(l+1)} \cdot a_j^{(l)}$$

chain rule

$$\delta_i^{(l)} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l)}} = \sum_k \underbrace{\frac{\partial L(y, \hat{y})}{\partial z_k^{(l+1)}}}_{\delta_k^{(l+1)}} \cdot \frac{\partial z_k^{(l+1)}}{\partial z_i^{(l)}}$$

$$L(y, \hat{y}) = y \log(\hat{y}) + (1 - y)\log(1 - \hat{y})$$

$$g(z) = \frac{1}{1 + e^{-z}} \qquad \delta_i^{(l+1)} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}}$$

Backprop

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g\left(z^{(2)}\right)$$

$$\vdots$$

$$a^{(l)} = g(z^{(l)})$$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g\left(z^{(l+1)}\right)$$

$$\vdots$$

$$\widehat{y} = a^{(L+1)}$$

$$\frac{\partial L(y, \widehat{y})}{\partial \Theta_{i,j}^{(l)}} = \frac{\partial L(y, \widehat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta_{i,j}^{(l)}} =: \delta_i^{(l+1)} \cdot a_j^{(l)}$$

$$\delta_i^{(l)} = \frac{\partial L(y, \widehat{y})}{\partial z_i^{(l)}} = \sum_k \frac{\partial L(y, \widehat{y})}{\partial z_k^{(l+1)}} \cdot \frac{\partial z_k^{(l+1)}}{\partial z_i^{(l)}}$$

$z^{(l+1)} \in \mathbb{R}^m$

chain rule

$$= \sum_k \delta_k^{(l+1)} \cdot \Theta_{k,i}^{(l)} \, g'(z_i^{(l)})$$

$$= a_i^{(l)}(1 - a_i^{(l)}) \sum_k \delta_k^{(l+1)} \cdot \Theta_{k,i}^{(l)}$$

$$L(y, \widehat{y}) = y \log(\widehat{y}) + (1 - y)\log(1 - \widehat{y})$$

$$g(z) = \frac{1}{1 + e^{-z}} \qquad \delta_i^{(l+1)} = \frac{\partial L(y, \widehat{y})}{\partial z_i^{(l+1)}}$$

Recursion from $L \to 1$

Backprop

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g\left(z^{(2)}\right)$$

$$\vdots$$

$$a^{(l)} = g(z^{(l)})$$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g\left(z^{(l+1)}\right)$$

$$\vdots$$

$$\widehat{y} = a^{(L+1)}$$

$$\frac{\partial L(y, \widehat{y})}{\partial \Theta^{(l)}_{i,j}} = \frac{\partial L(y, \widehat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta^{(l)}_{i,j}} =: \delta_i^{(l+1)} \cdot a_j^{(l)}$$

$$\delta_i^{(l)} = a_i^{(l)}(1 - a_i^{(l)}) \sum_k \delta_k^{(l+1)} \cdot \Theta^{(l)}_{k,i}$$

$$L(y, \widehat{y}) = y \log(\widehat{y}) + (1 - y)\log(1 - \widehat{y})$$

$$g(z) = \frac{1}{1 + e^{-z}} \qquad \delta_i^{(l+1)} = \frac{\partial L(y, \widehat{y})}{\partial z_i^{(l+1)}}$$

Backprop

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g\left(z^{(2)}\right)$$

$$\vdots$$

$$a^{(l)} = g(z^{(l)})$$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g\left(z^{(l+1)}\right)$$

$$\vdots$$

$$\hat{y} = a^{(L+1)}$$

$$\frac{\partial L(y, \hat{y})}{\partial \Theta_{i,j}^{(l)}} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta_{i,j}^{(l)}} =: \delta_i^{(l+1)} \cdot a_j^{(l)}$$

$$\delta_i^{(l)} = a_i^{(l)}(1 - a_i^{(l)}) \sum_k \delta_k^{(l+1)} \cdot \Theta_{k,i}^{(l)}$$

$$\delta_i^{(L+1)} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(L+1)}} = \frac{\partial}{\partial z_i^{(L+1)}} \left[y \log(g(z^{(L+1)})) + (1-y)\log(1 - g(z^{(L+1)})) \right]$$

calculus

$$= \frac{y}{g(z^{(L+1)})} g'(z^{(L+1)}) - \frac{1-y}{1 - g(z^{(L+1)})} g'(z^{(L+1)})$$

$$= y - g(z^{(L+1)}) = y - a^{(L+1)}$$

error-term

$$L(y, \hat{y}) = y \log(\hat{y}) + (1-y)\log(1 - \hat{y})$$

$$g(z) = \frac{1}{1 + e^{-z}} \qquad \delta_i^{(l+1)} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}}$$

Backprop

Θ_{ij}

$$\frac{\partial L(y,\hat{y})}{\partial \Theta_{i,j}^{(l)}} = \frac{\partial L(y,\hat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta_{i,j}^{(l)}} =: \delta_i^{(l+1)} \cdot a_j^{(l)}$$

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g\left(z^{(2)}\right)$$

$$\delta_i^{(l)} = a_i^{(l)}(1 - a_i^{(l)}) \sum_k \delta_k^{(l+1)} \cdot \Theta_{k,i}^{(l)}$$

$$\vdots$$

$$\delta^{(L+1)} = y - a^{(L+1)}$$

$$a^{(l)} = g(z^{(l)})$$

Recursive Algorithm!

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g\left(z^{(l+1)}\right)$$

$$\vdots$$

$$\hat{y} = a^{(L+1)}$$

$$L(y,\hat{y}) = y\log(\hat{y}) + (1-y)\log(1-\hat{y})$$

$$g(z) = \frac{1}{1+e^{-z}} \qquad \delta_i^{(l+1)} = \frac{\partial L(y,\hat{y})}{\partial z_i^{(l+1)}}$$

full batch GD

Backpropagation

$\mathrm{Loss}(\Theta) = \frac{1}{n}\sum_{i=1}^{n} \mathrm{Loss}(x_i, y_i)$

Set $\Delta_{ij}^{(l)} = 0 \quad \forall l, i, j$ (Used to accumulate gradient)

For each training instance $(\mathbf{x}_i, y_i)$: $t = 1, \ldots, n$

$\frac{\partial \mathrm{Loss}(\Theta)}{\partial \Theta}$

Set $\mathbf{a}^{(1)} = \mathbf{x}_i$

Compute $\{\mathbf{a}^{(2)}, \ldots, \mathbf{a}^{(L)}\}$ via forward propagation

Compute $\boldsymbol{\delta}^{(L)} = \mathbf{a}^{(L)} - y_i$

Compute errors $\{\boldsymbol{\delta}^{(L-1)}, \ldots, \boldsymbol{\delta}^{(2)}\}$ for x_t

Compute gradients $\Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

Gradient

Compute avg regularized gradient $D_{ij}^{(l)} = \begin{cases} \frac{1}{n}\Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} & \text{if } j \neq 0 \\ \frac{1}{n}\Delta_{ij}^{(l)} & \text{otherwise} \end{cases}$

$\Theta \leftarrow \Theta - \eta D$

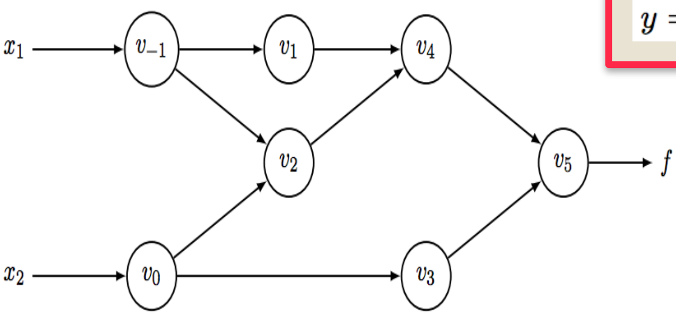
Naive $O(L^2)$

backprop $O(L)$

Autodiff

Backprop for this simple network architecture is a special case of ***reverse-mode auto-differentiation:***

check wiki



$$\boxed{y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - \sin(x_2)}$$

Forward Primal Trace		
$v_{-1} = x_1$		$= 2$
$v_0 = x_2$		$= 5$
$v_1 = \ln v_{-1}$		$= \ln 2$
$v_2 = v_{-1} \times v_0$		$= 2 \times 5$
$v_3 = \sin v_0$		$= \sin 5$
$v_4 = v_1 + v_2$		$= 0.693 + 10$
$v_5 = v_4 - v_3$		$= 10.693 + 0.959$
$y = v_5$		$= 11.652$

Reverse Adjoint (Derivative) Trace			
$\bar{\boldsymbol{x}}_1$	$= \bar{\boldsymbol{v}}_{-1}$		$\mathbf{= 5.5}$
$\bar{\boldsymbol{x}}_2$	$= \bar{\boldsymbol{v}}_0$		$\mathbf{= 1.716}$
$\bar{v}_{-1}$	$= \bar{v}_{-1} + \bar{v}_1 \frac{\partial v_1}{\partial v_{-1}}$	$= \bar{v}_{-1} + \bar{v}_1 / v_{-1}$	$= 5.5$
$\bar{v}_0$	$= \bar{v}_0 + \bar{v}_2 \frac{\partial v_2}{\partial v_0}$	$= \bar{v}_0 + \bar{v}_2 \times v_{-1}$	$= 1.716$
$\bar{v}_{-1}$	$= \bar{v}_2 \frac{\partial v_2}{\partial v_{-1}}$	$= \bar{v}_2 \times v_0$	$= 5$
$\bar{v}_0$	$= \bar{v}_3 \frac{\partial v_3}{\partial v_0}$	$= \bar{v}_3 \times \cos v_0$	$= -0.284$
$\bar{v}_2$	$= \bar{v}_4 \frac{\partial v_4}{\partial v_2}$	$= \bar{v}_4 \times 1$	$= 1$
$\bar{v}_1$	$= \bar{v}_4 \frac{\partial v_4}{\partial v_1}$	$= \bar{v}_4 \times 1$	$= 1$
$\bar{v}_3$	$= \bar{v}_5 \frac{\partial v_5}{\partial v_3}$	$= \bar{v}_5 \times (-1)$	$= -1$
$\bar{v}_4$	$= \bar{v}_5 \frac{\partial v_5}{\partial v_4}$	$= \bar{v}_5 \times 1$	$= 1$
$\bar{v}_5$	$= \bar{y}$	$= 1$	

This is the special sauce in Tensorflow, PyTorch, Theano, …