

Lecture 14 – Backpropagation & Intro to Convolutional neural networks

Prof. Maggie Makar

Announcements


- This week's quiz is a midterm reflection. Tentative due date is Sunday
- Project 2 will be released on Wednesday 3/13. Due Tuesday 3/26 at 10 pm.
 - Get started early
 - Really. Get started early
 - Quickstart on 3/14 at 7:00pm in DOW 1014
 - Important note: get started early

Feedback from class evaluation

- Send mmakar@umich.edu an email:
 - Subject line: [introducing-myself]
 - Include a recent picture of yourself
 - Include how you spell your name phonetically in English
 - (optional) include how you spell your name in your native language if other than English
- I have posted/will be posting full lectures
 - One cautionary note
- More real-life examples/applications
- 120 minutes without a break is too long – need a short break!
 - Send a question suggestion or youtube video suggestion here:
<https://bit.ly/eecs445-gimme-a-break> (or scan QR code)
- “Going to Office hours is a huge help for this class as its very complex, its always helpful to see other students in the same boat and to be able to bounce ideas around to get the correct solution” and “The discussion sections are super helpful in order to learn how exactly we have to apply all the equations we learn in class.”
- Request for more office hours



Class outline

- Recap: Setup and challenges in training NNs
 - Error backpropagation
 - Updating weights in the last layer
 - Updating weights in the first layer
 - Insight from three-layer NN updates
 - Practical considerations
 - Motivating CNNs
 - 2D Convolution and zero padding
 - Max Pooling
 - The final layer
- 

Training neural networks: Setup

- Training data

$$S_n = \{(\bar{x}^{(i)}, y^{(i)})\}_{i=1}^n$$

$$\bar{x} \in \mathbb{R}^d \quad y \in \{-1, +1\}$$

- Neural network

$$\bar{\theta} = [w_{10}^{(1)}, w_{11}^{(1)}, \dots, w_{JK}^{(L)}]$$

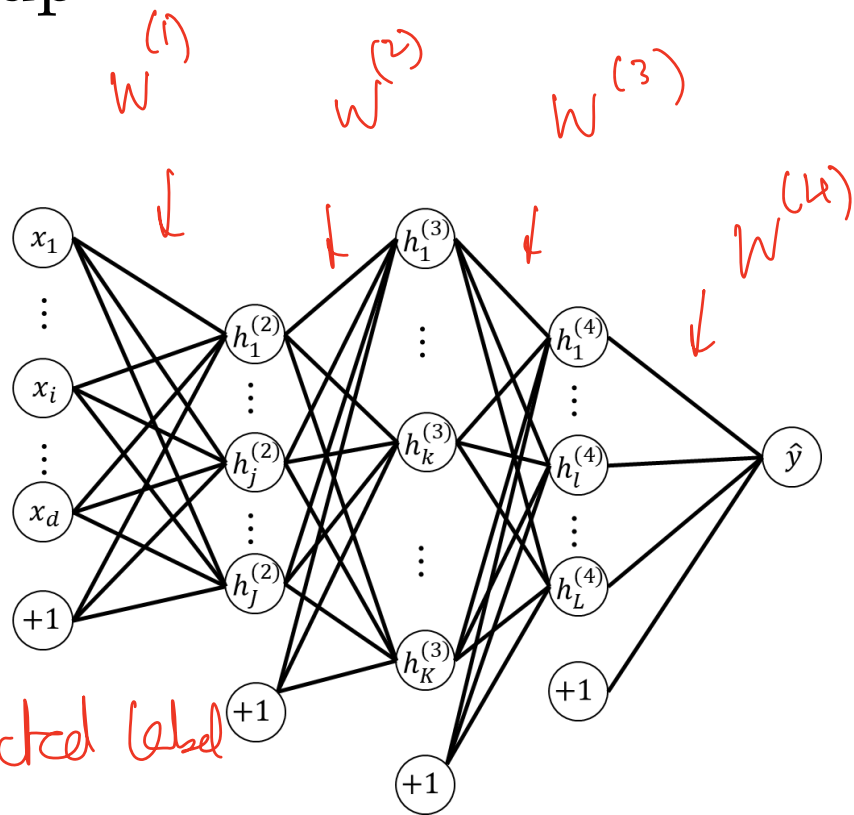
$$\underline{h(\bar{x}; \bar{\theta})} = \underline{z^{(L+1)}} = z_1^{(L+1)}$$

target label

- Loss function

$$J(\bar{\theta}) = \frac{1}{n} \sum_{i=1}^n \text{Loss}(y^{(i)}, h(\bar{x}^{(i)}; \bar{\theta}))$$

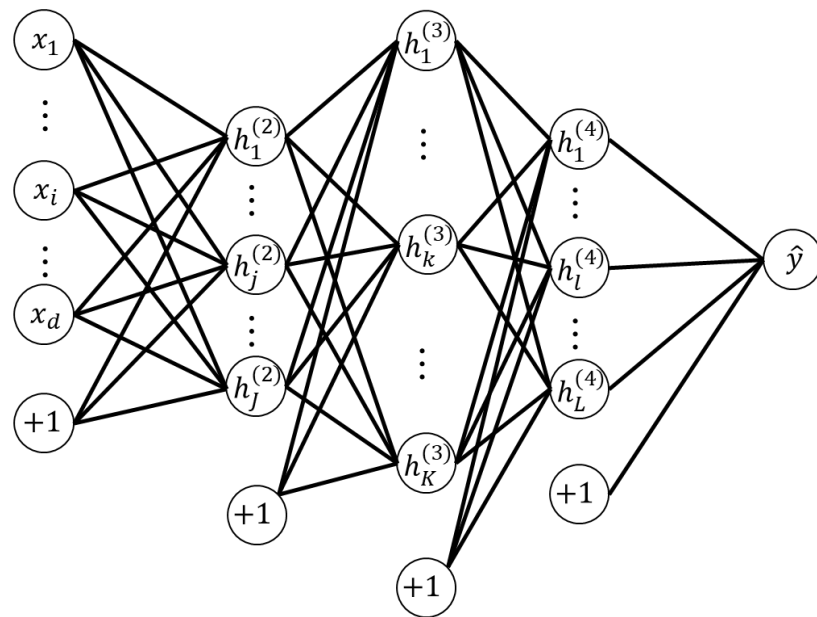
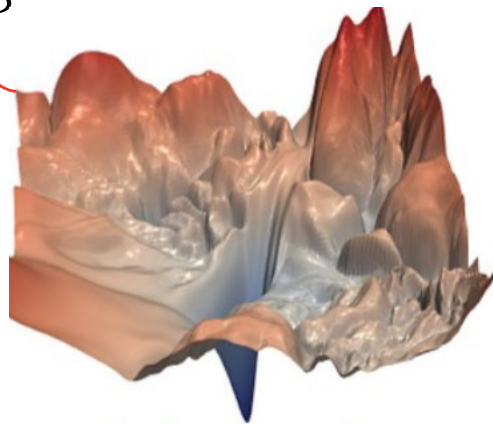
predicted label



Training neural networks: challenges

1. Non-convex loss
2. Typically uses large datasets
- 3.

No closed form solution
SGD ~~FGD~~

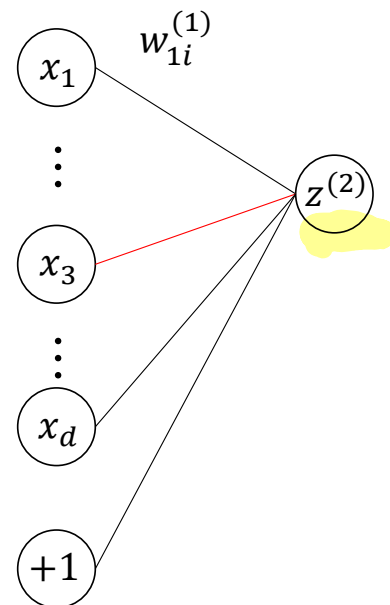


Is SGD enough? Single layer updates

- Single layer NN with no non-linearities, and hinge loss
- Goal: get parameter updates for all $\bar{\theta} = [w_{10}^{(1)}, w_{11}^{(1)}, \dots, w_{1d}^{(1)}]$
- Focus on one component of $\bar{\theta}$. E.g., $w_{13}^{(1)}$

$$\begin{aligned} \text{Loss}(y, h(\bar{x}; \bar{\theta})) &= \max\{1 - y h(\bar{x}; \bar{\theta}), 0\} \\ &= \max\{1 - y z^{(2)}, 0\} \\ &= \max\{1 - y (\sum_i^d \underbrace{w_{1i}^{(1)} x_i}_{\text{red arrow}} + w_{10}^{(1)}), 0\} \end{aligned}$$

- Derivative wrt to $w_{13}^{(1)}$
- $$\frac{\partial \text{Loss}(y, h(\bar{x}, \bar{\theta}))}{\partial w_{13}^{(1)}} = \begin{cases} -yx_3, & \text{if } 1 - yz^{(2)} > 0 \\ 0 & \text{otherwise} \end{cases}$$



- Update for the $k+1$ SGD iteration:

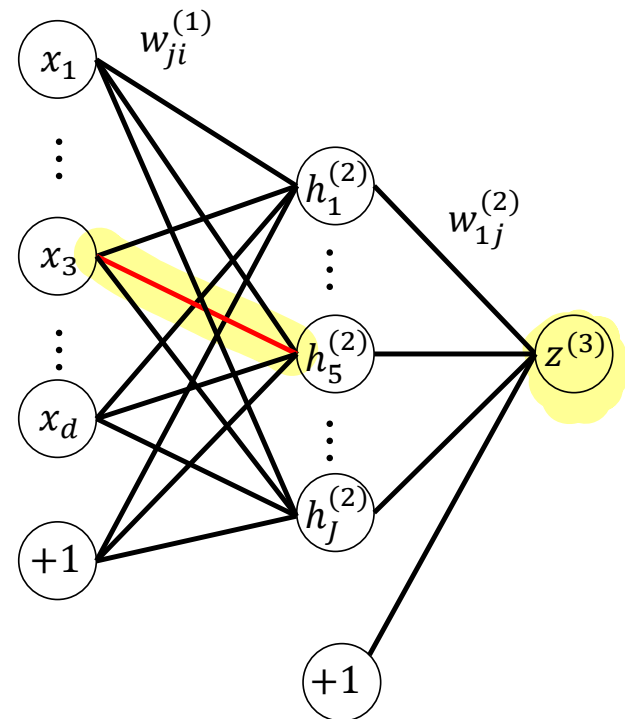
very $w_{13}^{(1,k+1)} = w_{13}^{(1,k)} + \eta_k y x_3 \mathbb{I}[1 - y z^{(2)} > 0]$

SGD iteration.

Is SGD enough? Two layer updates

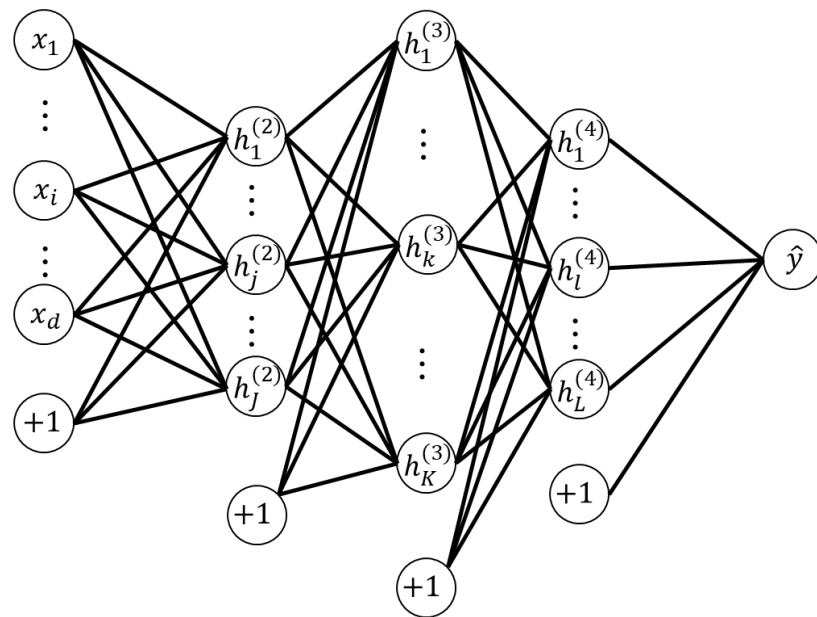
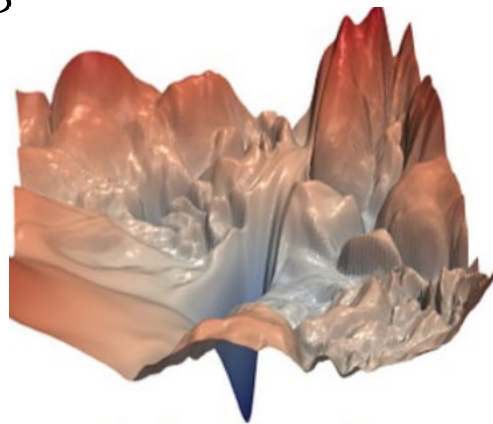
- Two layer NN with ReLU nonlinearities and hinge loss
- Goal: get parameter updates for all $\bar{\theta} = [w_{10}^{(1)}, \dots, w_{jd}^{(1)}, w_{10}^{(2)}, \dots, w_{1j}^{(2)}]$
- Focus on one component of $\bar{\theta}$. E.g., $w_{53}^{(1)}$

$$\begin{aligned}
 \text{Loss}(y, h(\bar{x}; \bar{\theta})) &= \max\{1 - y h(\bar{x}; \bar{\theta}), 0\} \\
 &= \max\{1 - y z^{(3)}, 0\} \\
 &= \max\{1 - y \left(\sum_{j=1}^J w_{1j}^{(2)} h_j^{(2)} + w_{10}^{(2)} \right), 0\} \\
 &= \max\{1 - y \left(\sum_{j=1}^J w_{1j}^{(2)} g(z_j^{(2)}) + w_{10}^{(2)} \right), 0\} \\
 &= \max\{1 - y \left(\sum_{j=1}^J w_{1j}^{(2)} \max\{z_j^{(2)}, 0\} + w_{10}^{(2)} \right), 0\} \\
 &= \max\{1 - y \left(\sum_{j=1}^J w_{1j}^{(2)} \max\left\{ \sum_{i=1}^d w_{ji}^{(1)} x_i + w_{j0}^{(1)}, 0 \right\} + w_{10}^{(2)} \right), 0\}
 \end{aligned}$$



Training neural networks: challenges

1. Non-convex loss
2. Typically uses large datasets
3. “Nested” functions make parameter updates complicated



TL;DPA: In principle, we can just use SGD to get the optimal parameters $\bar{\theta}$, but the SGD parameter updates would be exceedingly complex especially as the NN gets deeper. We need something to fix that

Training neural networks: Error backpropagation

Backprop = SGD + applying the chain rule

1. Initialize weights at small random values
2. Sample one data point
3. Update the weights as follows

$$\bar{\theta}^{(k+1)} = \bar{\theta}^{(k)} - \eta_k \nabla_{\bar{\theta}} \text{Loss}(y^{(i)}, h(\bar{x}; \bar{\theta}))$$

Chain rule:

$$\nabla_{\bar{\theta}} f(g(\bar{\theta})) = f'(g(\bar{\theta}))g'(\bar{\theta})$$

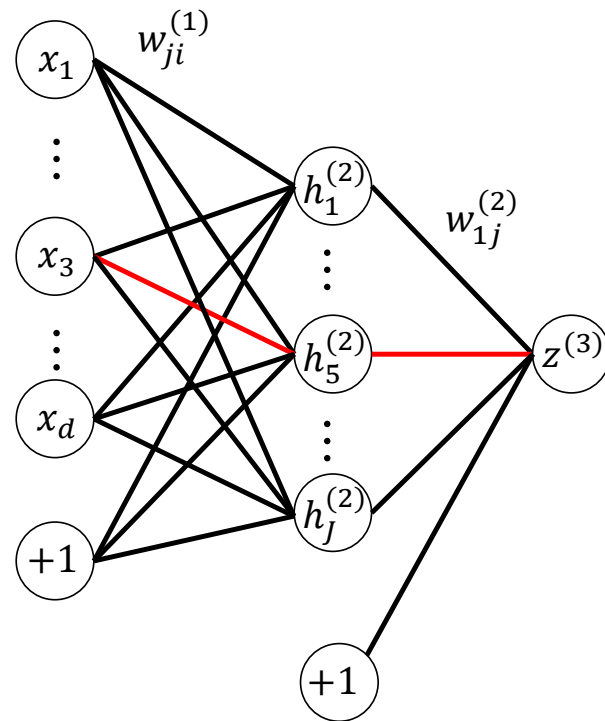
$$\frac{\partial f(g(\bar{\theta}))}{\partial \bar{\theta}} = \underbrace{\frac{\partial f(g(\bar{\theta}))}{\partial g(\bar{\theta})}}_{\frac{\delta f(\bar{\theta})}{\delta g(\bar{\theta})}} \underbrace{\frac{\partial g(\bar{\theta})}{\partial \bar{\theta}}}_{\frac{\delta g(\bar{\theta})}{\delta \bar{\theta}}}$$

Error backprop

- Two layer NN with ReLU nonlinearities and hinge loss
- Goal: get parameter updates for all $\bar{\theta} = [w_{10}^{(1)}, \dots, w_{jd}^{(1)}, w_{10}^{(2)}, \dots, w_{1j}^{(2)}]$
- Focus on two components of $\bar{\theta}$, one from each layer. E.g., $w_{15}^{(2)}$ $w_{53}^{(1)}$
- Need to get

$$\frac{\partial \text{Loss}(y, h(\bar{x}; \bar{\theta}))}{\partial w_{15}^{(2)}}, \frac{\partial \text{Loss}(y, h(\bar{x}; \bar{\theta}))}{\partial w_{53}^{(1)}}$$

- Use **Error Backprop**: it starts from the back end and “attributes” error to each of the parameters as you go towards the input layer. Think of Telephone, the game.



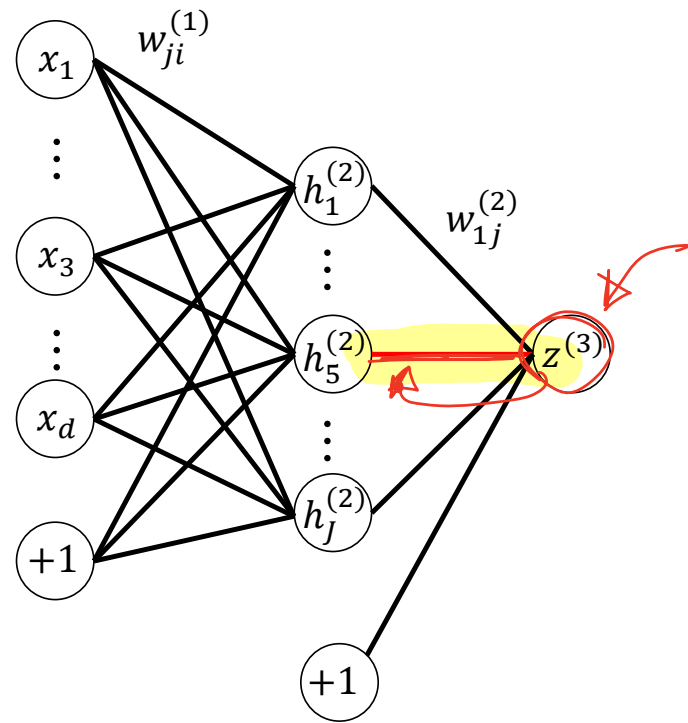
Error backprop: update for $w_{15}^{(2)}$

- Goal: estimate $\frac{\partial \text{Loss}(y, h(\bar{x}; \bar{\theta}))}{\partial w_{15}^{(2)}}$

- Using the chain rule:

$$\begin{aligned}\frac{\partial \text{Loss}(y, h(\bar{x}; \bar{\theta}))}{\partial w_{15}^{(2)}} &= \frac{\partial \text{Loss}(y, z^{(3)})}{\partial w_{15}^{(2)}} \\ &= \frac{\partial \text{Loss}(y, z^{(3)})}{\partial z^{(3)}} \cdot \frac{\partial z^{(3)}}{\partial w_{15}^{(2)}} \\ &= \frac{\partial \max\{1 - yz^{(3)}, 0\}}{\partial z^{(3)}} \cdot \frac{\partial \sum_{j=1}^J w_{1j}^{(2)} h_j^{(2)} + w_{10}^{(2)}}{\partial w_{15}^{(2)}} \\ &= -y \mathbb{I}[1 - yz^{(3)} > 0] \cdot h_5^{(2)} \\ &= -yh_5^{(2)} \mathbb{I}[1 - yz^{(3)} > 0]\end{aligned}$$

- Update $w_{15}^{(2)}$ as we typically do for SGD

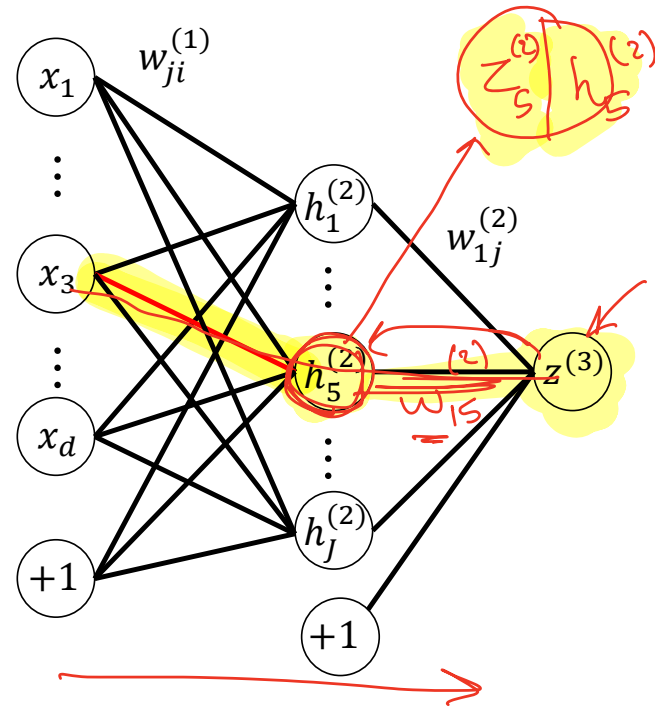


Error backprop: update for $w_{53}^{(1)}$

- Goal: estimate $\frac{\partial \text{Loss}(y, z^{(3)})}{\partial w_{53}^{(1)}}$
- Using the chain rule:

$$\begin{aligned}
 \frac{\partial \text{Loss}(y, z^{(3)})}{\partial w_{53}^{(1)}} &= \frac{\partial \text{Loss}(y, z^{(3)})}{\partial z^{(3)}} \cdot \frac{\partial z^{(3)}}{\partial h_5^{(2)}} \cdot \frac{\partial h_5^{(2)}}{\partial z_5^{(2)}} \cdot \frac{\partial z_5^{(2)}}{\partial w_{53}^{(1)}} \\
 &= \frac{\partial \max\{1 - yz^{(3)}, 0\}}{\partial z^{(3)}} \cdot \frac{\partial \sum_j w_{1j}^{(2)} h_j^{(2)} + w_{10}^{(2)}}{\partial h_5^{(2)}} \cdot \frac{\partial \max\{z_5^{(2)}, 0\}}{\partial z_5^{(2)}} \cdot \frac{\partial \sum_i w_{5i}^{(1)} x_i + w_{50}^{(1)}}{\partial w_{53}^{(1)}} \\
 &= -y \mathbb{I}[1 - yz^{(3)} > 0] \cdot w_{15}^{(2)} \cdot 1 \mathbb{I}[z_5^{(2)} > 0] \cdot x_3
 \end{aligned}$$

- Update $w_{53}^{(1)}$ as we typically do for SGD



Practical tips on getting derivative updates

- First derivative: numerator is always the loss
- Last derivative: denominator is always the parameter that we want to update
- Every numerator should be a direct function of the denominator
- Helpful: draw the path from the last ~~year~~ *layer* to the parameter to update & work backwards

Backpropagation to compute the gradient

• *shuffle the dataset*

• During the $k + 1$ iteration:

- Sample some \bar{x}, y from the data

- Make a prediction $\hat{y} = h(\bar{x}; \bar{\theta}^{(k)}) \rightarrow$ *Forward Pass.*

- Measure the loss of the prediction \hat{y} with respect to the true label y
Call that $\text{Loss}(y, h(\bar{x}; \bar{\theta}^{(k)}))$

↓

- Go through each node in reverse order to figure out the contribution of each node to the loss *Backward Pass*

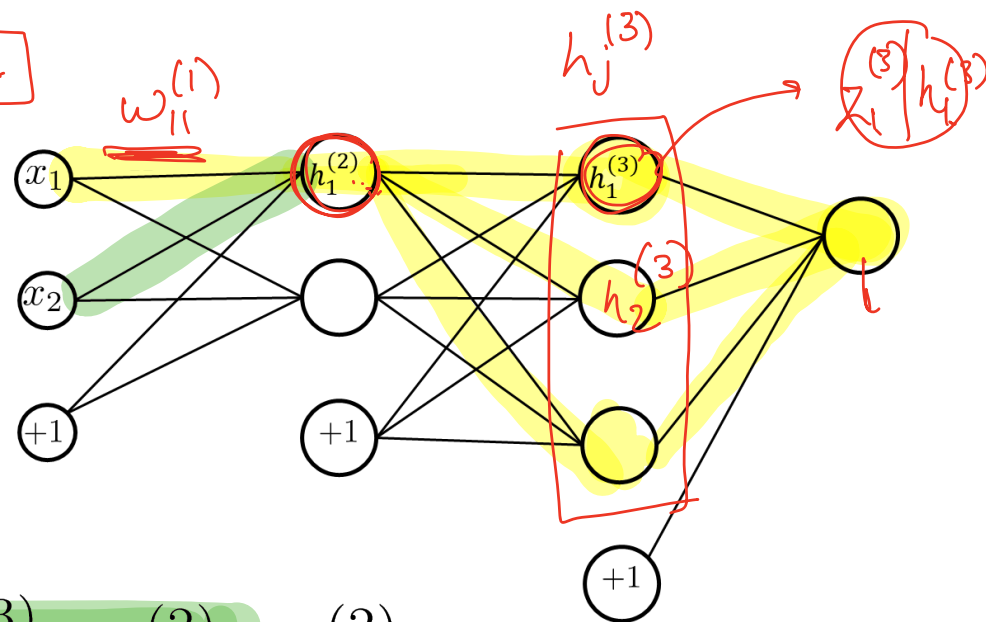
- To get $\bar{\theta}^{(k+1)}$, change the values of the parameters to reduce error (SGD step)

Three layer NN

$$\bar{\Theta} = [\underline{w_{11}^{(1)}} \dots]$$

fixed

$$w_{12}^{(1)}$$



$$\frac{\partial L}{\partial w_{11}^{(1)}} = \frac{\partial L}{\partial \hat{y}} \sum_{j=1}^3 \frac{\partial \hat{y}}{\partial h_j^{(3)}} \frac{\partial h_j^{(3)}}{\partial z_j^{(3)}} \frac{\partial z_j^{(3)}}{\partial h_1^{(2)}} \frac{\partial h_1^{(2)}}{\partial z_1^{(2)}} \frac{\partial z_1^{(2)}}{\partial w_{11}^{(1)}}$$

$$\frac{\partial L}{\partial w_{12}^{(1)}} = \frac{\partial L}{\partial \hat{y}} \sum_{j=1}^3 \frac{\partial \hat{y}}{\partial h_j^{(3)}} \frac{\partial h_j^{(3)}}{\partial z_j^{(3)}} \frac{\partial z_j^{(3)}}{\partial h_1^{(2)}} \frac{\partial h_1^{(2)}}{\partial z_1^{(2)}} \frac{\partial z_1^{(2)}}{\partial w_{12}^{(1)}}$$

TL;DPA:

1. To train NNs, we will use error backpropagation which is SGD + the chain rule
2. Using the chain rule makes taking derivatives easier and reduces the number of computations that we need to make (due to shared components in the derivative updates)

Training neural networks in practice

Architecture

```
class NeuralNet(nn.Module):  
    def __init__(self):  
        super(NeuralNet, self).__init__()  
        # hidden layer 1  
        self.fc1 = nn.Linear(10, 100)  
        # hidden layer 2  
        self.fc2 = nn.Linear(100, 50)  
        # hidden layer 3  
        self.fc3 = nn.Linear(50, 120)  
        # final output layer  
        self.final = nn.Linear(120, 1)  
  
    def forward(self, x):  
        x = F.relu(self.fc1(x))  
        x = F.relu(self.fc2(x))  
        x = F.relu(self.fc3(x))  
        x = F.relu(self.final(x))  
        return x
```

forward pass

```
# initialize your network  
neuralnet = NeuralNet()  
  
# create your optimizer  
optimizer = optim.SGD(neuralnet.parameters(), lr=0.01)  
  
# The following goes in your training loop:  
# reset gradients  
optimizer.zero_grad()  
output = neuralnet(x_i)  
loss = yourlossfunction(output, y_i)  
  
loss.backward() # calculates the derivatives  
optimizer.step() # does the update
```

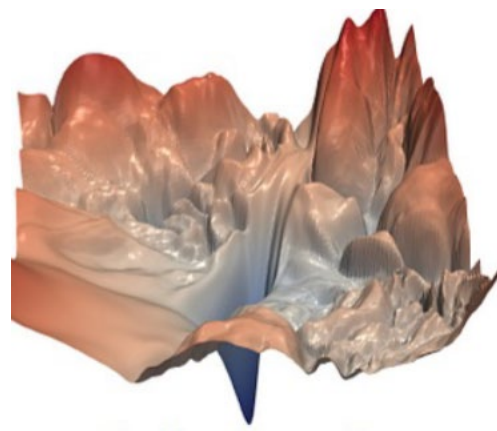
taking deriv

- Pytorch/tensorflow/JAX, etc uses an automatic differentiation method to calculate the backprop components

$$\theta^{(k+1)} = \theta^{(k)} - \eta \partial$$

Other practical considerations

- Recall the loss is non-convex
 - SGD does not guarantee getting to the global optimum
- The specifics of the optimizer can affect train and test error
 - Initialization, learning rate (dynamic/fixed)
- Fancier variants of SGD are typically used
 - ADAM, AdaGrad, etc
- Typically we use batch GD not SGD
 - Batch size is yet another design choice
- General advice: start from existing code



This is your 2 minute break!

- Turn to the people around you and discuss what was the last show you binge watched?
- Submit your own break questions or YouTube videos to this Google form



Introduction to Convolutional Neural Networks (CNNs)

Aka CNNs

Aka ConvNets

Image recognition and more

JAMA | **Original Investigation** | INNOVATIONS IN HEALTH CARE DELIVERY

Development and Validation of a Deep Learning Algorithm for Detection of Diabetic Retinopathy in Retinal Fundus Photographs

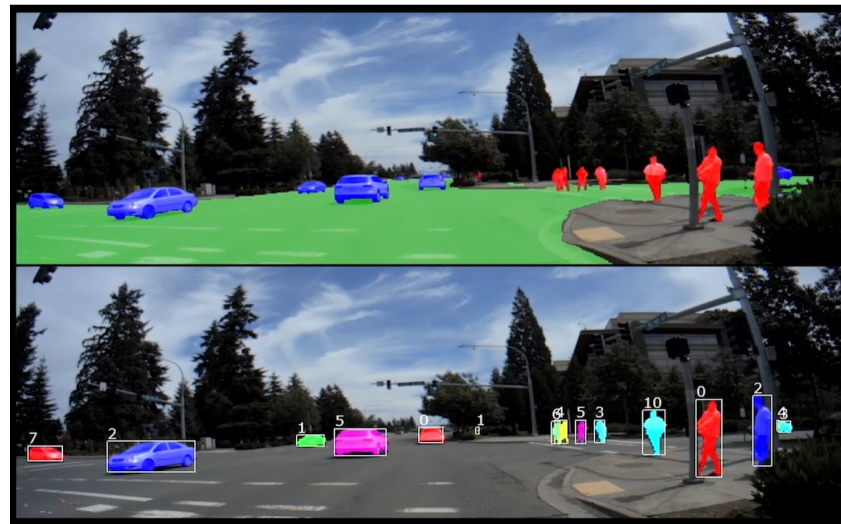
Varun Gulshan, PhD; Lily Peng, MD, PhD; Marc Coram, PhD; Martin C. Stumpe, PhD; Derek Wu, BS; Arunachalam Narayanaswamy, PhD; Subhashini Venugopalan, MS; Kasumi Widner, MS; Tom Madams, MEng; Jorge Cuadros, OD, PhD; Ramasamy Kim, OD, DNB; Rajiv Raman, MS, DNB; Philip C. Nelson, BS; Jessica L. Mega, MD, MPH; Dale R. Webster, PhD



Healthy



Referrable DR



Fundus images from Son et al, 2021

Autonomous vehicle demo from <https://blogs.nvidia.com/blog/2019/10/23/drive-labs-panoptic-segmentation/>

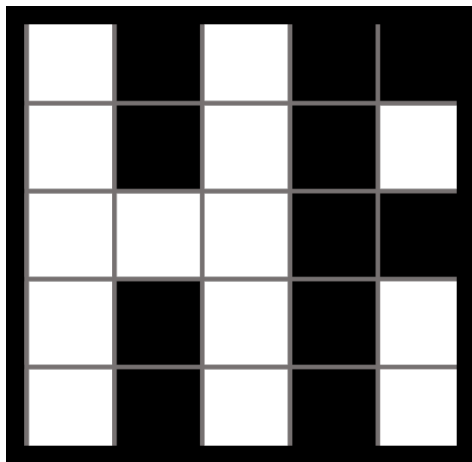
DTW image from <https://tinyurl.com/mvsfz75r>

Images as inputs



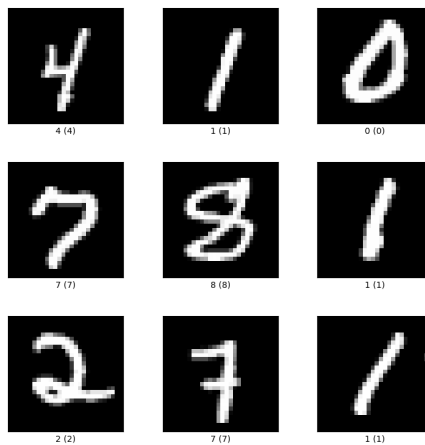
- Start with greyscale images

Images as inputs

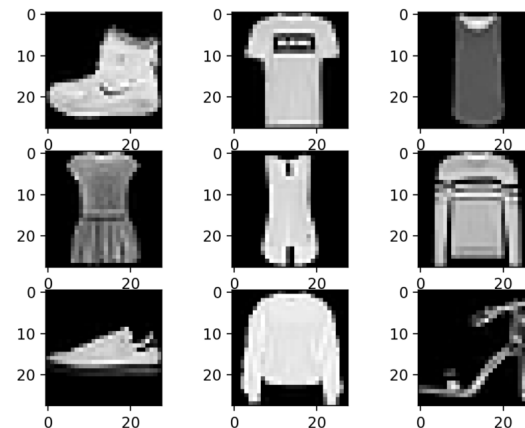


1	0	1	0	0
1	0	1	0	1
1	1	1	0	0
1	0	1	0	1
1	0	1	0	1

- Start with greyscale images
- Each pixel take a value between 0 and 1
- 0: black, 1: white



MNIST



Fashion MNIST