Deep Learning Basics
Role of Deep learning in Intelligent Vehicles

Feed from vehicle mounted sensors

Observing the surroundings: Multiple cameras and Lidars

Observing the cabin: Multiple cameras and IR sensors
Role of Deep learning in Intelligent Vehicles

Aptitude test long answer Q1:

**Step 1:** Detect Vehicles

**Step 2:** Track Vehicles
**Step 4:** Predict Intent and future locations of vehicles

**Step 5:** Determine if the predicted trajectories interact with the ego-vehicle’s planned path.

It would be intractable to hard code any of these steps!
Key idea: Use parametric function approximators for performing these steps

Detection

Inputs: $X$ Candidate bounding boxes

Outputs: $Y$ 0/1 labels

Inputs: $X$ Selected bounding box

Outputs: $Y$ deviation in location and shape of bounding box
Role of Deep learning in Intelligent Vehicles

Tracking

Inputs $X$: Vehicle $(i)$ currently tracked and detection $(j)$ in a new frame

$F_W(X)$

Output $Y$: $S(i,j)$ similarity score

Scores then used in bipartite matching [1]
Two types of models typically used:

**Classification models:** discrete outputs, typically probabilities

- Gaze zone estimation [2]
- Maneuver classification [3]
- Semantic segmentation [4]

**Regression models:** Continuous outputs

- Head pose estimation [5]
- Trajectory prediction [6]
- Similarity score estimation for tracking
Toy example: Two class linear classifier

\[ Y = \text{sigm}(w_1 x_1 + w_2 x_2 + w_0) \]

\[ \text{sigm}(x) = \frac{1}{1+\exp(-x)} \]
Linear classifier fail case:

\[ Y = \text{sigm}(w_1 x_1 + w_2 x_2 + w_0) \]
\[ \text{sigm}(x) = \frac{1}{1+\exp(-x)} \]

- Our linear classifier can’t handle this case

- **Idea**: Embed to 3-D and see if a plane can separate the points
**Idea:** Embed to 3-D and see if a plane can separate the points

Apply linear (actually affine) transformation to $\mathbf{x}$ to get 3-D vector $\mathbf{h}$:

$$\mathbf{h} = W\mathbf{x} + c$$

$$Y = \text{sigm}(w_1h_1 + w_2h_2 + w_3h_3 + w_0)$$
We need a non-linear function to separate out the points:

$$h = f(Wx + c)$$

- Typical non-linearities $f()$ used:
  - ReLu = $\max(0,x)$
  - Sigmoid
  - tanh
Neural Networks

- Deep neural networks have multiple stacked hidden layers

- Multiclass classification at output using softmax function:
  \[ x = [x_1, x_2, ..., x_n] \]
  \[ \text{softmax}(x_i) = \frac{\exp(x_i)}{\exp(x_1) + \exp(x_2) + ... + \exp(x_n)} \]

- Outputs can be interpreted as probabilities for each class
We now need to find a good value for our parameters $W$
We need a measure of *goodness (or badness)* of a particular value of $W$

- **Loss function** $L(f_W(X), Y)$ captures how *badly* the chosen set of parameters fit the true value of $Y$ for a *training sample*

- Typical loss functions:
  - Classification, Cross-entropy: $\sum_c Y_c \log(f_W(X_c))$
  - Regression, Mean-squared error: $|Y-f_W(X)|^2$

- **Training samples** are pre-labeled data points for which both $X$ and $Y$ are known: $D = [(X^{(1)}, Y^{(1)}), (X^{(2)}, Y^{(2)}), \ldots, (X^{(n)}, Y^{(n)})]$

- We can use this to find the optimal set of model parameters using:
  \[ W^* = \arg\min_W \sum_i L(f_W(X^{(i)}), Y^{(i)}) \]
• **Training objective:**
  \[ W^* = \arg\min_W \sum L(f_W(X^{(i)}),Y^{(i)}) \]

• We find \( W^* \) by using gradient descent on the objective defined above
  \[ W_{t+1} = W_t - \lambda \nabla L_{Wt} \]

• **Speeding up gradient descent:** In practice, the above update rule can be extremely slow and memory inefficient
  • Use smarter update rules that keep memory of past gradient values (eg. Momentum, Adam etc.)
  • Use *minibatches* of data to update gradients, rather than the entire dataset
Termination condition for gradient descent:

- A fixed number of iterations
- When the loss value stops changing*

The second condition needs a validation dataset due to overfitting
Deep learning models are powerful function approximators

**General approach:**

1. Collect labeled data $D = \{(X^{(1)},Y^{(1)}), (X^{(2)},Y^{(2)}), \ldots, (X^{(n)},Y^{(n)})\}$. Split into disjoint train ($D_{Tr}$), validation ($D_{Val}$) and test ($D_{Test}$) sets.
2. Define a *loss function*: $L(F_W(X), Y)$
3. Define model structure
4. Set hyperparameters
5. Search for parameters $W$ that minimize $L$ on $D_{Tr}$
6. Evaluate model using $D_{Val}$
7. Test model using $D_{Test}$
Gradient computation

For small models we could hand calculate the gradients of

$$L(W) = \sum_i L(f_w(X^{(i)}), Y^{(i)})$$

But things can quickly get out of hand for more complex models [7]:
Gradient computation w/ Backpropagation

To methodically calculate gradients, we need to view our models as computational graphs:

Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

Slides borrowed from CS231n: Convolutional Neural Networks for Visual Recognition
Fei-Fei Li & Justin Johnson & Serena Yeung[1]
Gradient computation w/ Backpropagation

Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, \ y = 5, \ z = -4 \)

\[
q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1
\]

\[
f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q
\]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)

Slides borrowed from CS231n: Convolutional Neural Networks for Visual Recognition
Fei-Fei Li & Justin Johnson & Serena Yeung
Gradient computation w/ Backpropagation

Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
Gradient computation w/ Backpropagation

Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, \ y = 5, \ z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \ \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \ \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \ \frac{\partial f}{\partial y}, \ \frac{\partial f}{\partial z} \)
Gradient computation w/ Backpropagation

Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)

Chain rule:
\[
\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \cdot \frac{\partial q}{\partial y}
\]
Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)

Chain rule:

\[ \frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \cdot \frac{\partial q}{\partial x} \]

Slides borrowed from CS231n: Convolutional Neural Networks for Visual Recognition
Fei-Fei Li & Justin Johnson & Serena Yeung
Gradient computation w/ Backpropagation

Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

\( \text{e.g. } x = -2, \ y = 5, \ z = -4 \)

\[
q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1
\]

\[
f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q
\]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)

Chain rule:

\[
\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}
\]
Gradient computation with Backpropagation

Slides borrowed from CS231n: Convolutional Neural Networks for Visual Recognition
Fei-Fei Li & Justin Johnson & Serena Yeung
Gradient computation w/ Backpropagation

Slides borrowed from CS231n: Convolutional Neural Networks for Visual Recognition
Fei-Fei Li & Justin Johnson & Serena Yeung
Gradient computation with Backpropagation

\[ \frac{\partial z}{\partial x}, \frac{\partial z}{\partial y}, \frac{\partial L}{\partial z} \]

“local gradient”

Slides borrowed from CS231n: Convolutional Neural Networks for Visual Recognition
Fei-Fei Li & Justin Johnson & Serena Yeung
Gradient computation w/ Backpropagation

Slides borrowed from CS231n: Convolutional Neural Networks for Visual Recognition
Fei-Fei Li & Justin Johnson & Serena Yeung
Another example:

\[ f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \]
Gradient computation w/ Backpropagation

Another example:

\[ f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \]

\[
\begin{align*}
    f(x) &= e^x \
    f_a(x) &= ax
\end{align*}
\]

\[
\begin{align*}
    \frac{df}{dx} &= e^x \
    \frac{df}{dx} &= a
\end{align*}
\]

\[
\begin{align*}
    f(x) &= \frac{1}{x} \
    f_c(x) &= c + x
\end{align*}
\]

\[
\begin{align*}
    \frac{df}{dx} &= -\frac{1}{x^2} \
    \frac{df}{dx} &= 1
\end{align*}
\]

Slides borrowed from CS231n: Convolutional Neural Networks for Visual Recognition
Fei-Fei Li & Justin Johnson & Serena Yeung
Another example:

\[
 f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}
\]

\[
 f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x
\]

\[
 f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a
\]

\[
 f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2
\]

\[
 f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1
\]
Gradient computation w/ Backpropagation

Another example:

\[ f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \]

\[
\begin{align*}
  f(x) &= e^x \\
  f_a(x) &= ax
\end{align*}
\]

\[
\begin{align*}
  \frac{df}{dx} &= e^x \\
  \frac{df}{dx} &= a
\end{align*}
\]

\[
\begin{align*}
  f(x) &= \frac{1}{x} \\
  f_c(x) &= c + x
\end{align*}
\]

\[
\begin{align*}
  \frac{df}{dx} &= -\frac{1}{x^2} \\
  \frac{df}{dx} &= 1
\end{align*}
\]

Slides borrowed from CS231n: Convolutional Neural Networks for Visual Recognition
Fei-Fei Li & Justin Johnson & Serena Yeung
Another example:

\[ f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \]

\[
\begin{align*}
    f(x) &= e^x \\
    f_a(x) &= ax
\end{align*}
\]

\[
\begin{align*}
    \frac{df}{dx} &= e^x \\
    \frac{df}{dx} &= a
\end{align*}
\]

\[
\begin{align*}
    f(x) &= \frac{1}{x} \\
    f_c(x) &= c + x
\end{align*}
\]

\[
\begin{align*}
    \frac{df}{dx} &= -1/x^2 \\
    \frac{df}{dx} &= 1
\end{align*}
\]
Gradient computation w/ Backpropagation

Another example:

\[ f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \]

\[ f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x \]

\[ f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a \]

\[ f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -\frac{1}{x^2} \]

\[ f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1 \]

(1)(−0.53) = −0.53

Slides borrowed from CS231n: Convolutional Neural Networks for Visual Recognition
Fei-Fei Li & Justin Johnson & Serena Yeung
Another example:

\[ f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}} \]

\[ f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x \]

\[ f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a \]

\[ f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -\frac{1}{x^2} \]

\[ f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1 \]
Another example: 

\[ f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \]

\[ (e^{-1})(-0.53) = -0.20 \]

\[ f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x \]

\[ f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a \]

\[ f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -\frac{1}{x^2} \]

\[ f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1 \]
Gradient computation w/ Backpropagation

Another example: 

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$

$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -\frac{1}{x^2}$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$
Another example: \[ f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \]

\[
\begin{align*}
f(x) &= e^x & \Rightarrow & & \frac{df}{dx} &= e^x \\
f_a(x) &= ax & \Rightarrow & & \frac{df}{dx} &= a \\
\quad & f(x) = \frac{1}{x} & \Rightarrow & & \frac{df}{dx} &= -\frac{1}{x^2} \\
\quad & f_c(x) = c + x & \Rightarrow & & \frac{df}{dx} &= 1 
\end{align*}
\]

(-1) * (-0.20) = 0.20
Another example: 

\[ f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \]

\[ f(x) = e^x \quad \Rightarrow \quad \frac{df}{dx} = e^x \]

\[ f_a(x) = ax \quad \Rightarrow \quad \frac{df}{dx} = a \]

\[ f(x) = \frac{1}{x} \quad \Rightarrow \quad \frac{df}{dx} = -\frac{1}{x^2} \]

\[ f_c(x) = c + x \quad \Rightarrow \quad \frac{df}{dx} = 1 \]
Another example:

\[ f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \]

\[ f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x \]

\[ f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a \]

\[ f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1 \]

\[ f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -\frac{1}{x^2} \]

[local gradient] x [upstream gradient]

\[ [1] \times [0.2] = 0.2 \]

\[ [1] \times [0.2] = 0.2 \ (\text{both inputs!}) \]
Another example:

$$f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$$

**Equations:****

$$f(x) = e^x \quad \Rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \Rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \Rightarrow \quad \frac{df}{dx} = -\frac{1}{x^2}$$

$$f_c(x) = c + x \quad \Rightarrow \quad \frac{df}{dx} = 1$$

---

**Slides borrowed from CS231n: Convolutional Neural Networks for Visual Recognition**

Fei-Fei Li & Justin Johnson & Serena Yeung
Gradient computation w/ Backpropagation

Another example:

\[ f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}} \]

[local gradient] x [upstream gradient]

\[ x_0: [2] \times [0.2] = 0.4 \]
\[ w_0: [-1] \times [0.2] = -0.2 \]

\[ f(x) = e^x \quad \Rightarrow \quad \frac{df}{dx} = e^x \]
\[ f_a(x) = ax \quad \Rightarrow \quad \frac{df}{dx} = a \]
\[ f(x) = \frac{1}{x} \quad \Rightarrow \quad \frac{df}{dx} = -\frac{1}{x^2} \]
\[ f_c(x) = c + x \quad \Rightarrow \quad \frac{df}{dx} = 1 \]
Gradient computation w/ Backpropagation

\[ f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \]

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

\[ \frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1 + e^{-x})^2} = \left( \frac{1 + e^{-x} - 1}{1 + e^{-x}} \right) \left( \frac{1}{1 + e^{-x}} \right) = (1 - \sigma(x)) \sigma(x) \]

sigmoid function

sigmoid gate

Slides borrowed from CS231n: Convolutional Neural Networks for Visual Recognition
Fei-Fei Li & Justin Johnson & Serena Yeung
Gradient computation w/ Backpropagation

\[
f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}
\]

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

\[
\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1 + e^{-x})^2} = \left( \frac{1 + e^{-x} - 1}{1 + e^{-x}} \right) \left( \frac{1}{1 + e^{-x}} \right) = (1 - \sigma(x)) \sigma(x)
\]

sigmoid function

sigmoid gate

\[
(0.73) \times (1 - 0.73) = 0.2
\]
Patterns in backward flow

**add** gate: gradient distributor

**max** gate: gradient router

**mul** gate: gradient switcher

---

Slides borrowed from CS231n: Convolutional Neural Networks for Visual Recognition
Fei-Fei Li & Justin Johnson & Serena Yeung
Gradients add at branches
Gradient computation w/ Backpropagation

Gradients for vectorized code

\[ \frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial x} \]

(x, y, z are now vectors)

This is now the Jacobian matrix (derivative of each element of z w.r.t. each element of x)

“local gradient”

\[ \frac{\partial L}{\partial z} \]

\[ \frac{\partial z}{\partial y} \]

\[ \frac{\partial z}{\partial x} \]

f

y

x

y

z

Gradients

Slides borrowed from CS231n: Convolutional Neural Networks for Visual Recognition
Fei-Fei Li & Justin Johnson & Serena Yeung
A vectorized example: $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)^2_i$
A vectorized example: \( f(x, W) = \| W \cdot x \|^2 = \sum_{i=1}^{n} (W \cdot x)_i^2 \)

\( \in \mathbb{R}^n \quad \in \mathbb{R}^{n \times n} \)
A vectorized example: $f(x, W) = \|W \cdot x\|^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$
A vectorized example: \[ f(x, W) = \|W \cdot x\|^2 = \sum_{i=1}^{n} (W \cdot x)^2_i \]

\[
\begin{bmatrix}
0.1 & 0.5 \\
-0.3 & 0.8 \\
\end{bmatrix}
\begin{bmatrix}
0.2 \\
0.4 \\
\end{bmatrix}
\]

\[
W \quad \ast \\
\downarrow \quad \downarrow \\
L2 \\
\]

\[
q = W \cdot x = \begin{pmatrix}
W_{1,1}x_1 + \cdots + W_{1,n}x_n \\
\vdots \\
W_{n,1}x_1 + \cdots + W_{n,n}x_n \\
\end{pmatrix}
\]

\[
f(q) = \|q\|^2 = q_1^2 + \cdots + q_n^2
\]
Gradient computation w/ Backpropagation

A vectorized example: \[ f(x, W) = \|W \cdot x\|^2 = \sum_{i=1}^{n} (W \cdot x)^2_i \]

\[
W = \begin{bmatrix}
0.1 & 0.5 \\
-0.3 & 0.8
\end{bmatrix}
\]

\[
x = \begin{bmatrix}
0.2 \\
0.4
\end{bmatrix}
\]

\[
q = W \cdot x = \begin{pmatrix}
W_{1,1}x_1 + \cdots + W_{1,n}x_n \\
\vdots \\
W_{n,1}x_1 + \cdots + W_{n,n}x_n
\end{pmatrix}
\]

\[ f(q) = \|q\|^2 = q_1^2 + \cdots + q_n^2 \]
A vectorized example: 

\[
\begin{bmatrix}
0.1 & 0.5 \\
-0.3 & 0.8
\end{bmatrix}
\begin{bmatrix}
0.2 \\
0.4
\end{bmatrix}
= \begin{bmatrix}
0.22 \\
0.26
\end{bmatrix}
\]

\[
f(x, W) = \|W \cdot x\|^2 = \sum_{i=1}^{n} (W \cdot x)_i^2
\]

\[
f(q) = \|q\|^2 = q_1^2 + \cdots + q_n^2
\]

\[
\frac{\partial f}{\partial q_i} = 2q_i
\]

\[
\nabla_q f = 2q
\]
Gradient computation w/ Backpropagation

\[ f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)^2_i \]

A vectorized example:
\[
\begin{bmatrix}
0.1 & 0.5 \\
-0.3 & 0.8
\end{bmatrix}
\begin{bmatrix}
0.2 \\
0.4
\end{bmatrix}
\]

\[
q = W \cdot x = \begin{pmatrix}
W_{1,1}x_1 + \cdots + W_{1,n}x_n \\
\vdots \\
W_{n,1}x_1 + \cdots + W_{n,n}x_n
\end{pmatrix}
\]

\[ f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2 \]

\[ \frac{\partial f}{\partial q_i} = 2q_i \]

\[ \nabla_q f = 2q \]

Slides borrowed from CS231n: Convolutional Neural Networks for Visual Recognition
Fei-Fei Li & Justin Johnson & Serena Yeung
Gradient computation w/ Backpropagation

A vectorized example: \( f(x, W) = \|W \cdot x\|^2 = \sum_{i=1}^{n} (W \cdot x)_{i}^2 \)

\[
W = \begin{bmatrix}
0.1 & 0.5 \\
-0.3 & 0.8 \\
0.088 & 0.176 \\
0.104 & 0.208 \\
0.2 & 0.4 \\
\end{bmatrix}
\]

\[
x = \begin{bmatrix}
0.22 \\
0.26 \\
0.44 \\
0.52 \\
\end{bmatrix}
\]

\[
L2 = \begin{bmatrix}
0.116 \\
1.00 \\
\end{bmatrix}
\]

\[
q = W \cdot x = \begin{pmatrix}
W_{1,1}x_1 + \cdots + W_{1,n}x_n \\
\vdots \\
W_{n,1}x_1 + \cdots + W_{n,n}x_n
\end{pmatrix}
\]

\[
f(q) = \|q\|^2 = q_1^2 + \cdots + q_n^2
\]

\[
\nabla_W f = 2q \cdot x^T
\]

\[
\frac{\partial q_k}{\partial W_{i,j}} = 1_{k=i}x_j
\]

\[
\frac{\partial f}{\partial W_{i,j}} = \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial W_{i,j}}
\]

\[
= \sum_k (2q_k)(1_{k=i}x_j)
\]

\[
= 2q_i x_j
\]
A vectorized example: \[ f(x, W) = \|W \cdot x\|^2 = \sum_{i=1}^{n} (W \cdot x)_{i}^2 \]

\[
\begin{bmatrix}
0.1 & 0.5 \\
-0.3 & 0.8 \\
0.088 & 0.176 \\
0.104 & 0.208 \\
0.2 & 0.4 \\
-0.112 & 0.636
\end{bmatrix}
\begin{bmatrix}
0.22 \\
0.26 \\
0.44 \\
0.52 \\
0.116 \\
1.00
\end{bmatrix}
\]

\[ \nabla_x f = 2W^T \cdot q \]

\[
f(q) = \|q\|^2 = q_1^2 + \cdots + q_n^2
\]

\[
\frac{\partial q_k}{\partial x_i} = W_{k,i}
\]

\[
\frac{\partial f}{\partial x_i} = \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial x_i} = \sum_k 2q_k W_{k,i}
\]
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1

- **Input**: 1 x 3072
- **Weight**: 10 x 3072
- **Activation**: 1 x 10

1 number: the result of taking a dot product between a row of W and the input (a 3072-dimensional dot product)
Convolution Layer

32x32x3 image -> preserve spatial structure
Convolution Layer

32x32x3 image

5x5x3 filter

Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”
Convolutional Neural Networks

Convolution Layer

32x32x3 image

5x5x3 filter

Filters always extend the full depth of the input volume

Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”
Convolution Layer

32x32x3 image
5x5x3 filter \( w \)

**1 number:**
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. 5*5*3 = 75-dimensional dot product + bias)

\[ w^T x + b \]
Convolution Layer

A 32x32x3 image is convolved (slid) over all spatial locations with a 5x5x3 filter.

The result is an activation map with dimensions 28x28.
Convolutional Neural Networks

Convolution Layer

32x32x3 image
5x5x3 filter

convolve (slide) over all spatial locations

activation map

Slides borrowed from CS231n: Convolutional Neural Networks for Visual Recognition
Fei-Fei Li & Justin Johnson & Serena Yeung
Convolutional Neural Networks

Convolution Layer

Consider a second, green filter

32x32x3 image
5x5x3 filter

Convolve (slide) over all spatial locations

Activation maps

Slides borrowed from CS231n: Convolutional Neural Networks for Visual Recognition
Fei-Fei Li & Justin Johnson & Serena Yeung
For example, if we had 6 5x5 filters, we’ll get 6 separate activation maps:

We stack these up to get a “new image” of size 28x28x6!
**Preview:** ConvNet is a sequence of Convolutional Layers, interspersed with activation functions.

```
3       32
CONV, ReLU
 e.g. 6 5x5x3 filters
6       28
CONV, ReLU
 e.g. 10 5x5x6 filters
10      24
CONV, ReLU
```

...
Convolutional Neural Networks

[Zeiler and Fergus 2013]

Slides borrowed from CS231n: Convolutional Neural Networks for Visual Recognition
Fei-Fei Li & Justin Johnson & Serena Yeung
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter
A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter

=> 5x5 output
A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied with stride 2
Convolutional Neural Networks

A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied with stride 2
A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied with stride 2 => 3x3 output!
A closer look at spatial dimensions:

7

7x7 input (spatially) assume 3x3 filter applied with stride 3?
A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied with stride 3?

doesn’t fit! cannot apply 3x3 filter on 7x7 input with stride 3.
Convolutional Neural Networks

Output size:
\[(N - F) / \text{stride} + 1\]

e.g. \(N = 7, F = 3\):
- \(\text{stride 1} \Rightarrow (7 - 3)/1 + 1 = 5\)
- \(\text{stride 2} \Rightarrow (7 - 3)/2 + 1 = 3\)
- \(\text{stride 3} \Rightarrow (7 - 3)/3 + 1 = 2.33\)

Slides borrowed from CS231n: Convolutional Neural Networks for Visual Recognition
Fei-Fei Li & Justin Johnson & Serena Yeung
In practice: Common to zero pad the border

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

e.g. input 7x7
3x3 filter, applied with stride 1
pad with 1 pixel border => what is the output?

(recall:)
(N - F) / stride + 1

Slides borrowed from CS231n: Convolutional Neural Networks for Visual Recognition
Fei-Fei Li & Justin Johnson & Serena Yeung
In practice: Common to zero pad the border

e.g. input 7x7
3x3 filter, applied with \textbf{stride 1}
\textbf{pad with 1 pixel} border => what is the output?

7x7 output!
In practice: Common to zero pad the border

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

e.g. input 7x7
3x3 filter, applied with **stride 1**
**pad with 1 pixel** border => what is the output?

**7x7 output!**
in general, common to see CONV layers with
stride 1, filters of size $F \times F$, and zero-padding with
$(F-1)/2$. (will preserve size spatially)
e.g. $F = 3 \Rightarrow$ zero pad with 1
    $F = 5 \Rightarrow$ zero pad with 2
    $F = 7 \Rightarrow$ zero pad with 3
Examples time:

Input volume: $32 \times 32 \times 3$
10 5x5 filters with stride 1, pad 2

Output volume size: ?
Examples time:

Input volume: $32 \times 32 \times 3$
10 5x5 filters with stride 1, pad 2

Output volume size:
$(32 + 2 \times 2 - 5)/1 + 1 = 32$ spatially, so
$32 \times 32 \times 10$
Examples time:

Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?
Examples time:

Input volume: \(32 \times 32 \times 3\)
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?

Each filter has \(5 \times 5 \times 3 + 1 = 76\) params (+1 for bias)

\(\Rightarrow 76 \times 10 = 760\)
Pooling layer

- makes the representations smaller and more manageable
- operates over each activation map independently:

Slides borrowed from CS231n: Convolutional Neural Networks for Visual Recognition
Fei-Fei Li & Justin Johnson & Serena Yeung
MAX POOLING

Single depth slice

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

max pool with 2x2 filters and stride 2

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
Activation Functions

**Sigmoid**
\[ \sigma(x) = \frac{1}{1+e^{-x}} \]

**tanh**
\[ \tanh(x) \]

**ReLU**
\[ \max(0, x) \]

**Leaky ReLU**
\[ \max(0.1x, x) \]

**Maxout**
\[ \max(w_1^T x + b_1, w_2^T x + b_2) \]

**ELU**
\[ \begin{cases} 
  x & x \geq 0 \\
  \alpha(e^x - 1) & x < 0 
\end{cases} \]
Activation Functions

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

- Squashes numbers to range \([0, 1]\)
- Historically popular since they have nice interpretation as a saturating “firing rate” of a neuron

Sigmoid
Activation Functions

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

- Squashes numbers to range \([0, 1]\)
- Historically popular since they have nice interpretation as a saturating “firing rate” of a neuron

3 problems:

1. Saturated neurons “kill” the gradients
What happens when \( x = -10 \)?
What happens when \( x = 0 \)?
What happens when \( x = 10 \)?
Convolutional Neural Networks

Activation Functions

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

- Squashes numbers to range \([0,1]\)
- Historically popular since they have nice interpretation as a saturating “firing rate” of a neuron

3 problems:

1. Saturated neurons “kill” the gradients
2. Sigmoid outputs are not zero-centered

Slides borrowed from CS231n: Convolutional Neural Networks for Visual Recognition
Fei-Fei Li & Justin Johnson & Serena Yeung
Consider what happens when the input to a neuron is always positive...

\[ f \left( \sum_i w_i x_i + b \right) \]

What can we say about the gradients on \( w \)?
Always all positive or all negative :( (this is also why you want zero-mean data!)
Activation Functions

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

- Squashes numbers to range \([0, 1]\)
- Historically popular since they have nice interpretation as a saturating “firing rate” of a neuron

3 problems:

1. Saturated neurons “kill” the gradients
2. Sigmoid outputs are not zero-centered
3. \(\exp()\) is a bit compute expensive
Activation Functions

- Squashes numbers to range [-1, 1]
- zero centered (nice)
- still kills gradients when saturated :(

\[ \tanh(x) \]

[LeCun et al., 1991]
Convolutional Neural Networks

Activation Functions

- Computes $f(x) = \max(0, x)$
- Does not saturate (in $+$ region)
- Very computationally efficient
- Converges much faster than sigmoid/tanh in practice (e.g. 6x)
- Actually more biologically plausible than sigmoid

ReLU
(Rectified Linear Unit)

[Krizhevsky et al., 2012]
Activation Functions

- Computes $f(x) = \max(0, x)$
- Does not saturate (in $+$ region)
- Very computationally efficient
- Converges much faster than sigmoid/tanh in practice (e.g. 6x)
- Actually more biologically plausible than sigmoid

ReLU
(Rectified Linear Unit)

- Not zero-centered output
- An annoyance:

hint: what is the gradient when $x < 0$?

Slides borrowed from CS231n: Convolutional Neural Networks for Visual Recognition
Fei-Fei Li & Justin Johnson & Serena Yeung
What happens when $x = -10$?
What happens when $x = 0$?
What happens when $x = 10$?
Activation Functions

- Does not saturate
- Computationally efficient
- Converges much faster than sigmoid/tanh in practice! (e.g. 6x)
- will not “die”.

Leaky ReLU

\[ f(x) = \max(0.01x, x) \]
Activation Functions

Exponential Linear Units (ELU)

\[
f(x) = \begin{cases} 
  x & \text{if } x > 0 \\
  \alpha \left( \exp(x) - 1 \right) & \text{if } x \leq 0 
\end{cases}
\]

- All benefits of ReLU
- Closer to zero mean outputs
- Negative saturation regime compared with Leaky ReLU
  adds some robustness to noise

- Computation requires \( \exp() \)
Maxout “Neuron”
- Does not have the basic form of dot product -> nonlinearity
- Generalizes ReLU and Leaky ReLU
- Linear Regime! Does not saturate! Does not die!

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

Problem: doubles the number of parameters/neuron :(
TLDR: In practice:

- Use ReLU. Be careful with your learning rates
- Try out Leaky ReLU / Maxout / ELU
- Try out tanh but don’t expect much
- Don’t use sigmoid
Fully Connected Layer (FC layer)
- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks