Image Classification with CNN

ML Foundation

What we will learn today

- Optimization:
  - Gradient Descent
  - Backpropagation
- Neural Network
- Convolutional Neural Network

Image Classification

Loss Functions quantify preferences

Q: How do we find the best W ?

- We have some dataset of (x, y)
- We have a score function:
- We have a loss function:

$$s = f(x; W) = Wx$$

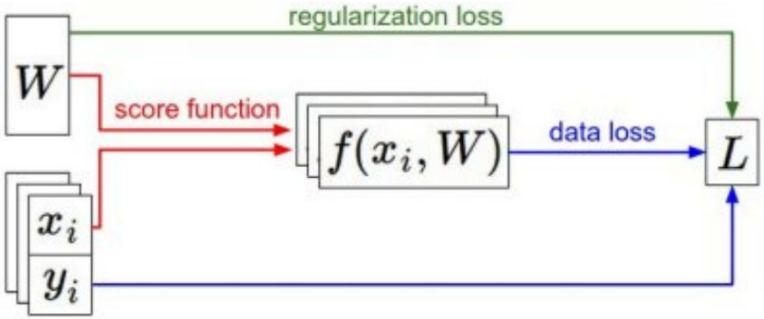
Linear classifier

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$ Softmax

SVM

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i + R(W)$$ Full loss



Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

How to train a Machine Learning Model

Step 1: Prepare dataset

Step 2: Model's weights(parameters are randomly initialize)

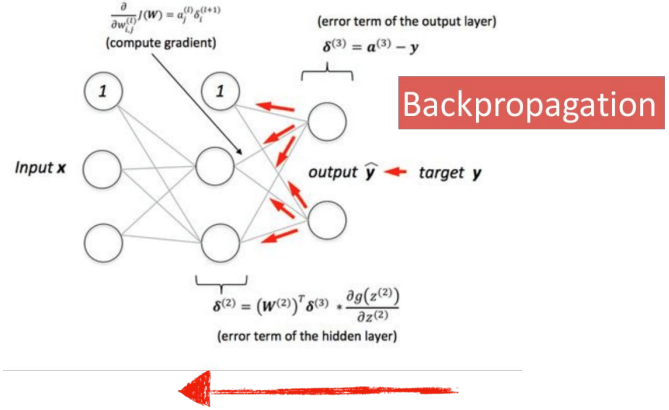
Step 3: Load training samples and feed into a model

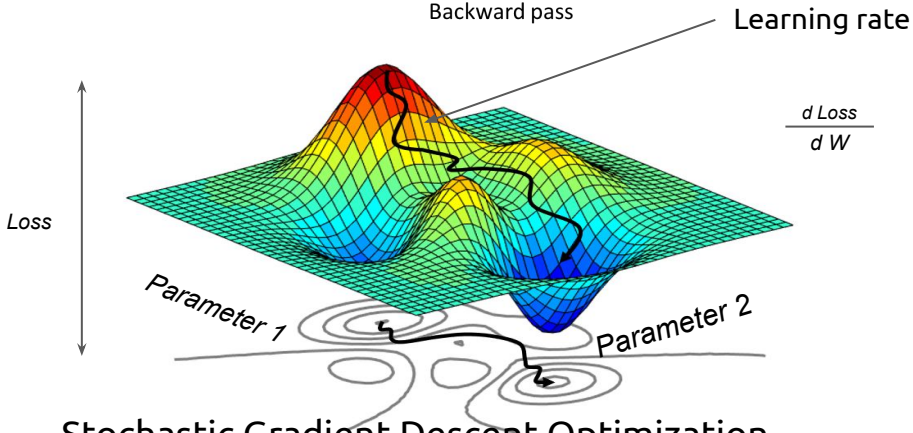
Step 4: A model make a prediction

Step 5: Compute loss (cost) function

Step 6: Parameters update (optimization)

Step 7: Validate the model using the validating dataset





Stochastic Gradient Descent Optimization

Optimization

We want to find a function (weights) that **achieve the lowest loss**.

$$W^* = \underset{w}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(f(x^{(i)}; W), y^{(i)})$$

$$W^* = \underset{W}{\operatorname{argmin}} J(W)$$

Loss function

$$f^* = \underset{f \in \mathcal{F}}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(f(x^{(i)}), y^{(i)})$$

Hypothesis space

Training data

Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Optimization

Idea #1: Random Search

15.5% accuracy! Not bad

But it is a **bad idea**.

```
# Assume X_test is [3073 x 10000], Y_test [10000 x 1]
scores = Wbest.dot(Xte_cols) # 10 x 10000, the class scores for all test examples
# find the index with max score in each column (the predicted class)
Yte_predict = np.argmax(scores, axis = 0)
# and calculation accuracy (fraction of predictions that are correct)
np.mean(Yte_predict = Yte)
# retrun 0.15555
```

Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Optimization

Idea #2: Follow the slop

Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Optimization

Gradient Descent

Optimization

Gradient Descent

In 1-dimension, the **derivative** of a function gives **the slope**:

$$\frac{df}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

In multiple dimensions, the **gradient** is the vector of (partial derivatives) along each dimension.

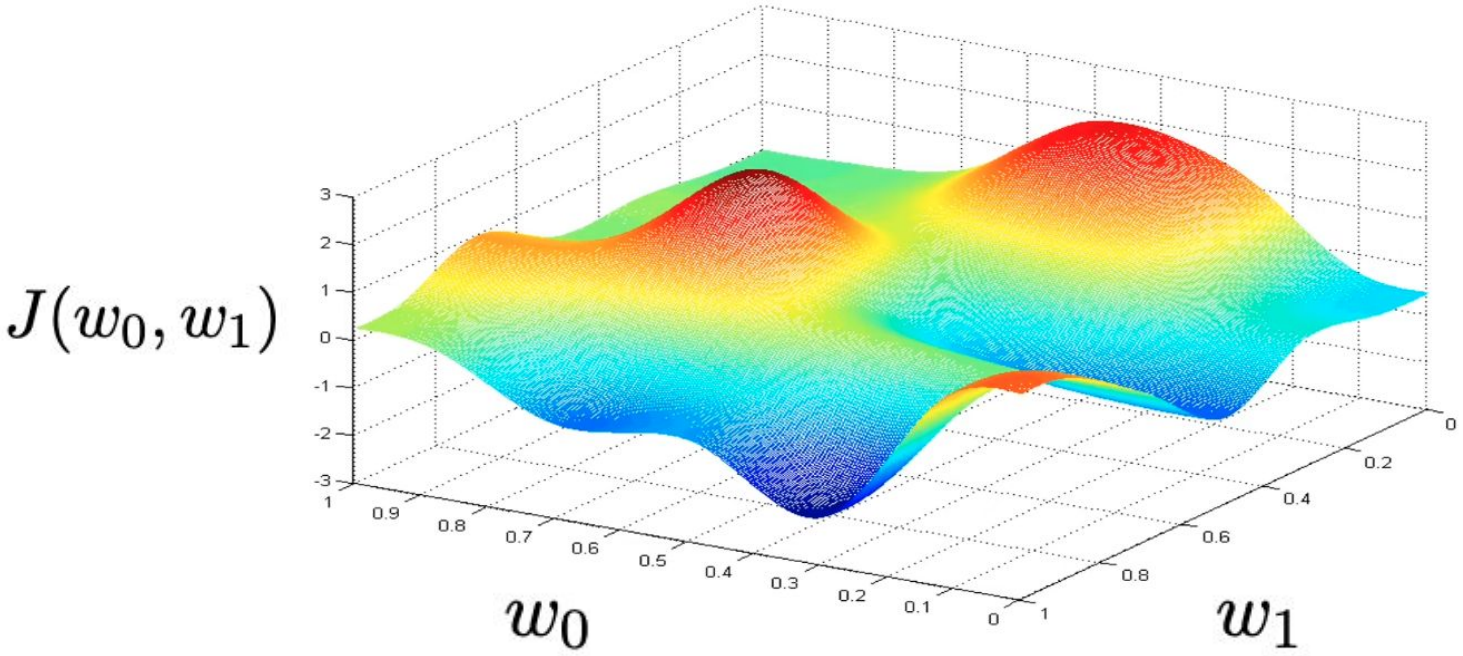
The **slope** in any **direction** is the **dot product** of the direction with the gradient. The direction of steepest descent is the **negative gradient**.

Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Optimization

Gradient Descent

$$W^* = \underset{W}{\operatorname{argmin}}\, J(W)$$

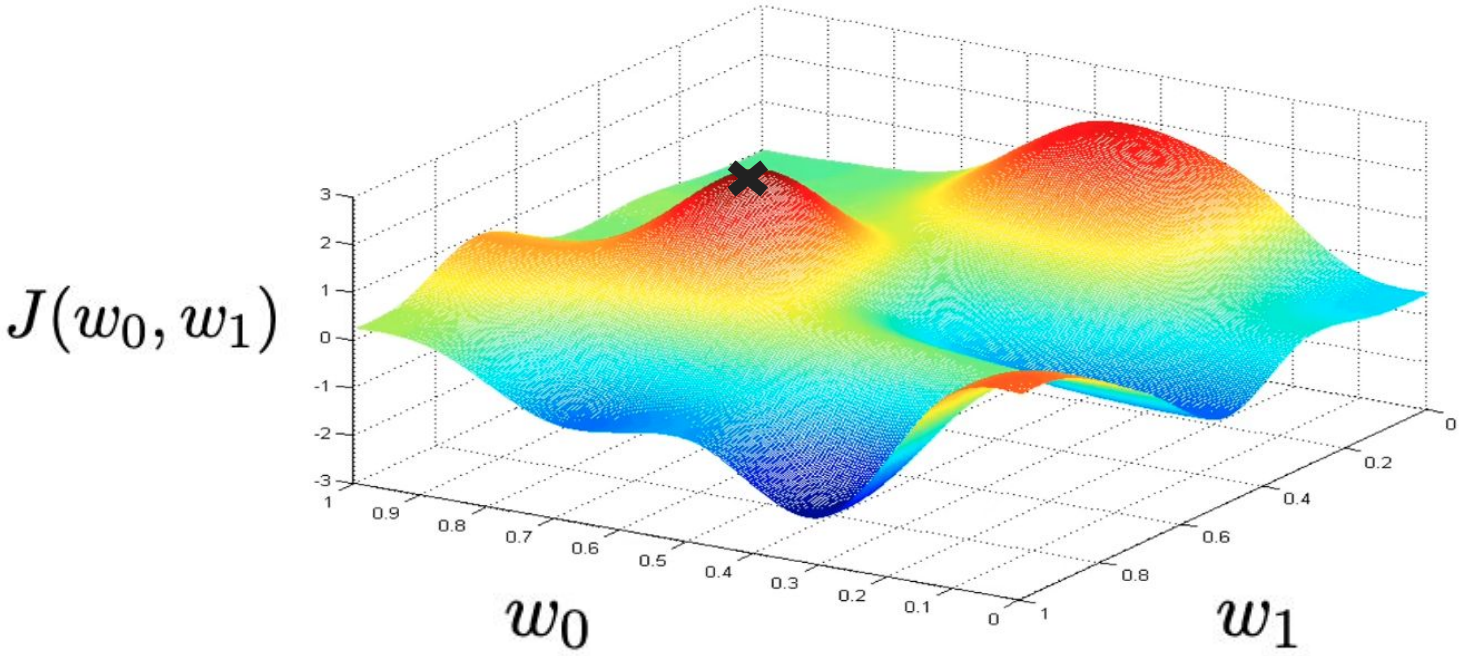


Slide adopted from: Alexander Amini, MIT 6.S191 Introduction to Deep Learning, MIT, http://introtodeeplearning.com/

Optimization

Gradient Descent

Random pick an initial (w_0, w_1)

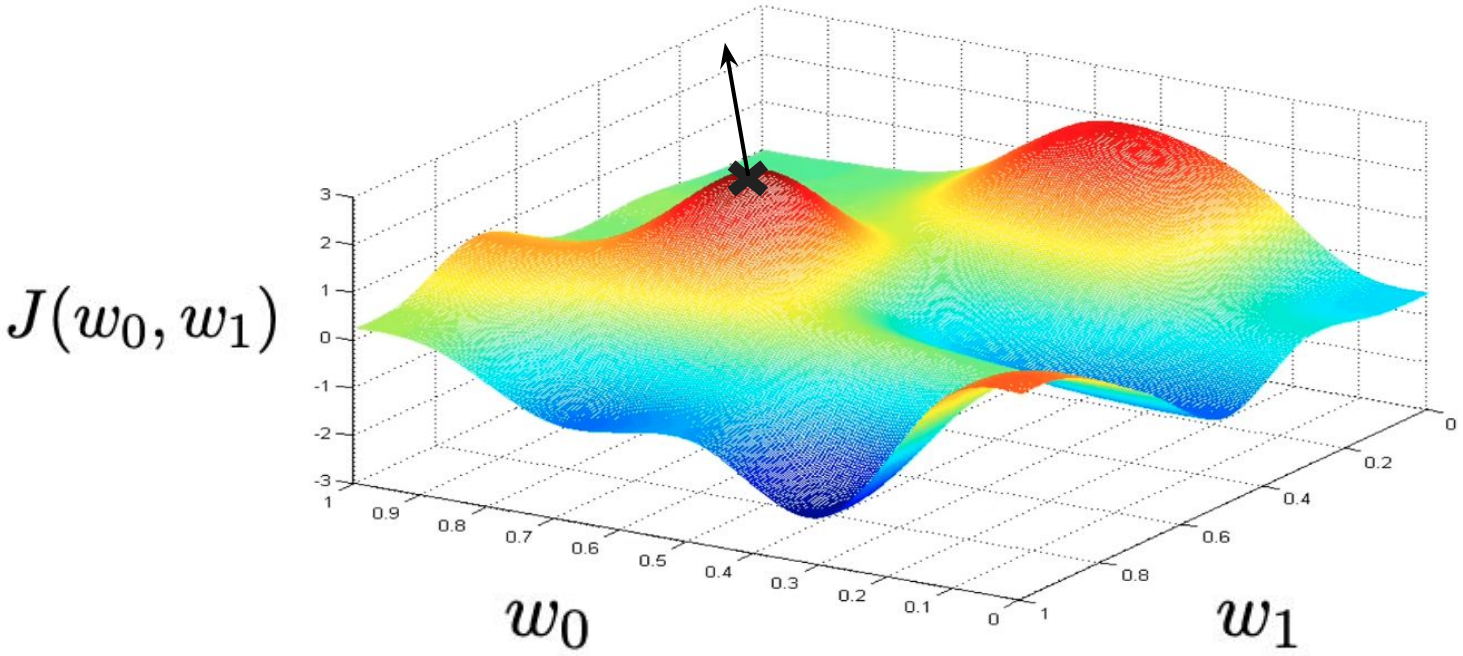


Slide adopted from: Alexander Amini, MIT 6.S191 Introduction to Deep Learning, MIT, http://introtodeeplearning.com/

Optimization

Gradient Descent

Compute gradient $\frac{\partial J(W)}{\partial W}$



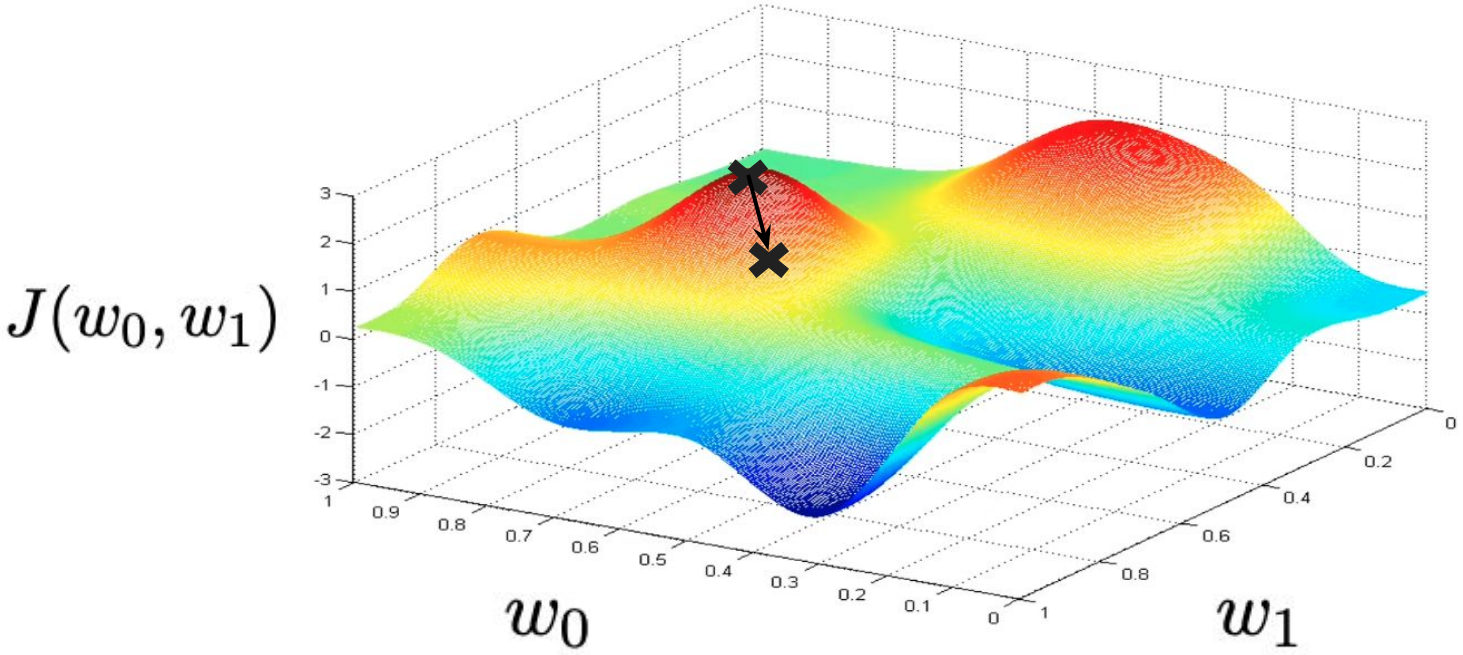
Slide adopted from: Alexander Amini, MIT 6.S191 Introduction to Deep Learning, MIT, http://introtodeeplearning.com/

Optimization

Gradient Descent

The direction of steepest descent is the **negative gradient**.

Take small step in opposite direction of gradient

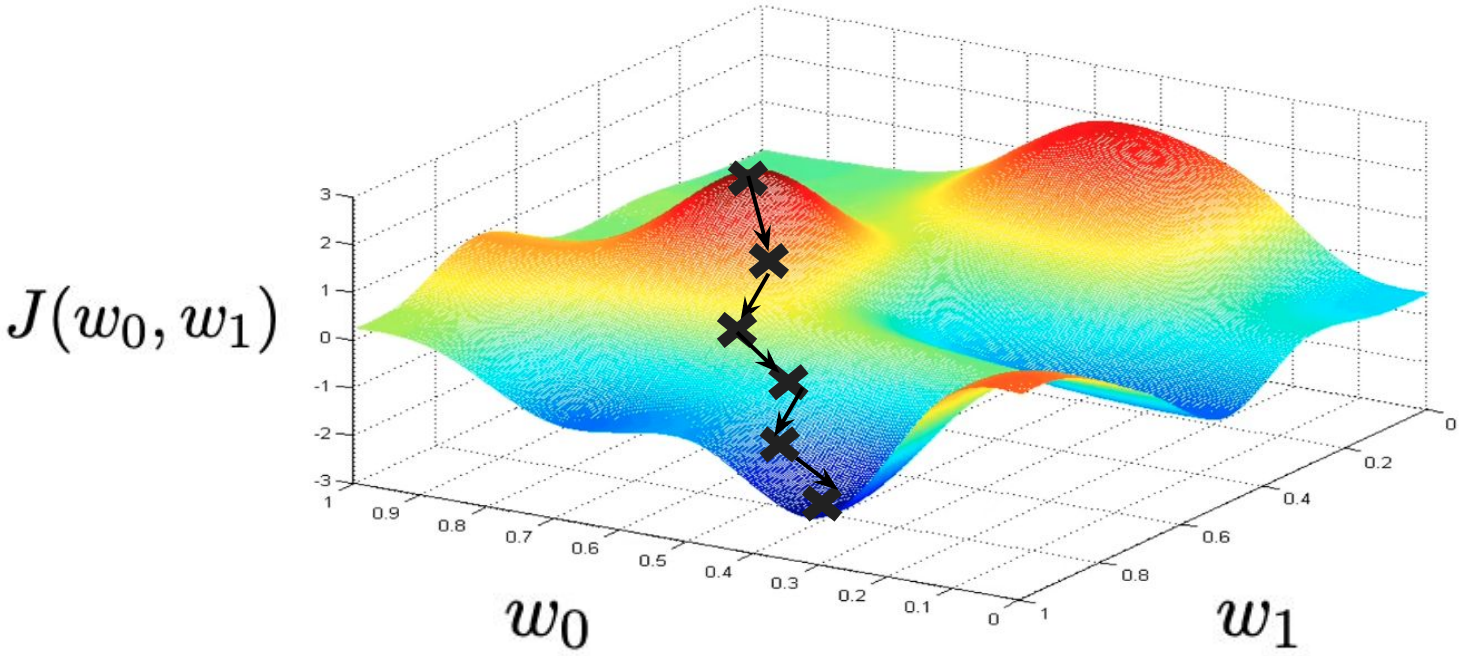


Slide adopted from: Alexander Amini, MIT 6.S191 Introduction to Deep Learning, MIT, http://introtodeeplearning.com/

Optimization

Gradient Descent

Repeat until convergence



Slide adopted from: Alexander Amini, MIT 6.S191 Introduction to Deep Learning, MIT, http://introtodeeplearning.com/

Optimization

Gradient Descent

Algorithm

1. Initialize weights randomly $\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
   a. Compute gradient $\frac{\partial J(W)}{\partial W}$
   b. Update weights $W \longleftarrow W - \eta \frac{\partial J(W)}{\partial W}$
3. Return weights

Slide adopted from: Alexander Amini, MIT 6.S191 Introduction to Deep Learning, MIT, http://introtodeeplearning.com/

Optimization

Gradient Descent

Algorithm

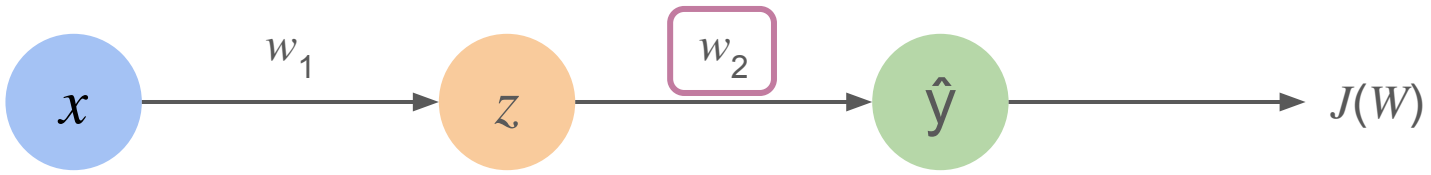
1. Initialize weights randomly $\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
   a. Compute gradient $\boxed{\frac{\partial J(W)}{\partial W}}$
   b. Update weights $W \longleftarrow W - \eta \frac{\partial J(W)}{\partial W}$
3. Return weights

Q: How do we compute this ?

Slide adopted from: Alexander Amini, MIT 6.S191 Introduction to Deep Learning, MIT, http://introtodeeplearning.com/

Optimization

Computing Gradients: Backpropagation

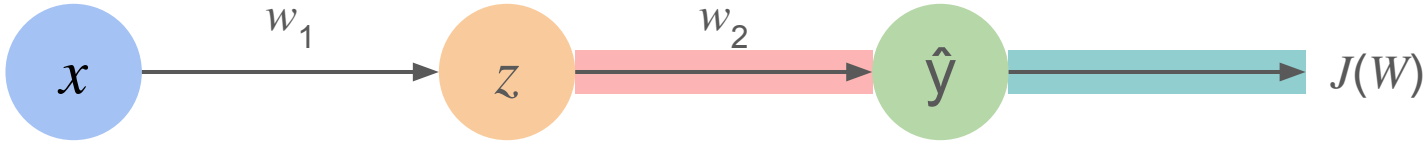


How does a small change in one weight (ex. w_2) affect the final loss ?

Slide adopted from: Alexander Amini, MIT 6.S191 Introduction to Deep Learning, MIT, http://introtodeeplearning.com/

Optimization

Computing Gradients: Backpropagation

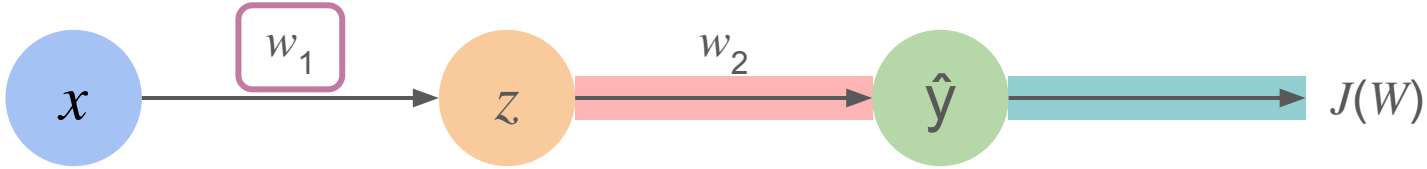


$$\frac{\partial J(W)}{\partial w_2} = \frac{\partial J(W)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_2}$$

Slide adopted from: Alexander Amini, MIT 6.S191 Introduction to Deep Learning, MIT, http://introtodeeplearning.com/

Optimization

Computing Gradients: Backpropagation



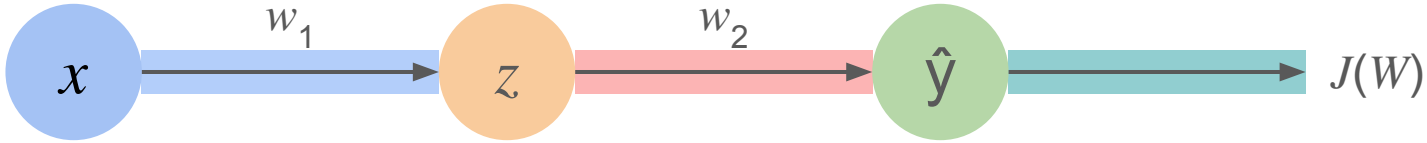
$$\frac{\partial J(W)}{\partial \boxed{w_1}} = \frac{\partial J(W)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_1}$$

Apply chain rule!

Slide adopted from: Alexander Amini, MIT 6.S191 Introduction to Deep Learning, MIT, http://introtodeeplearning.com/

Optimization

Computing Gradients: Backpropagation



$$\frac{\partial J(W)}{\partial w_1} = \frac{\partial J(W)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$

Repeat this for **every weight in the network** using gradients from later layers

Slide adopted from: Alexander Amini, MIT 6.S191 Introduction to Deep Learning, MIT, http://introtodeeplearning.com/

Optimization

Setting the Learning rate

$$W \longleftarrow W - \boxed{\eta} \frac{\partial J(W)}{\partial W}$$

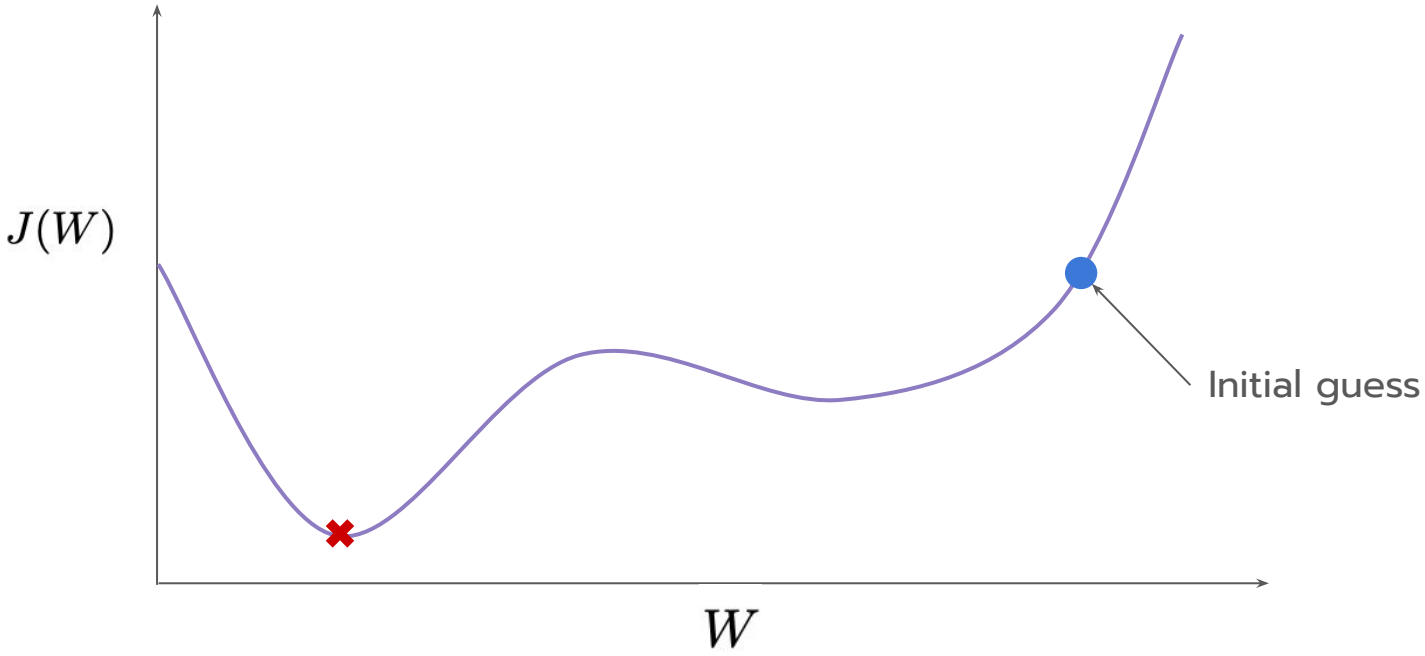
How can we set the **learning rate** ?

Slide adopted from: Alexander Amini, MIT 6.S191 Introduction to Deep Learning, MIT, http://introtodeeplearning.com/

Optimization

Setting the Learning rate

Small learning rate converges <u>slowly and gets stuck</u> in false local minima

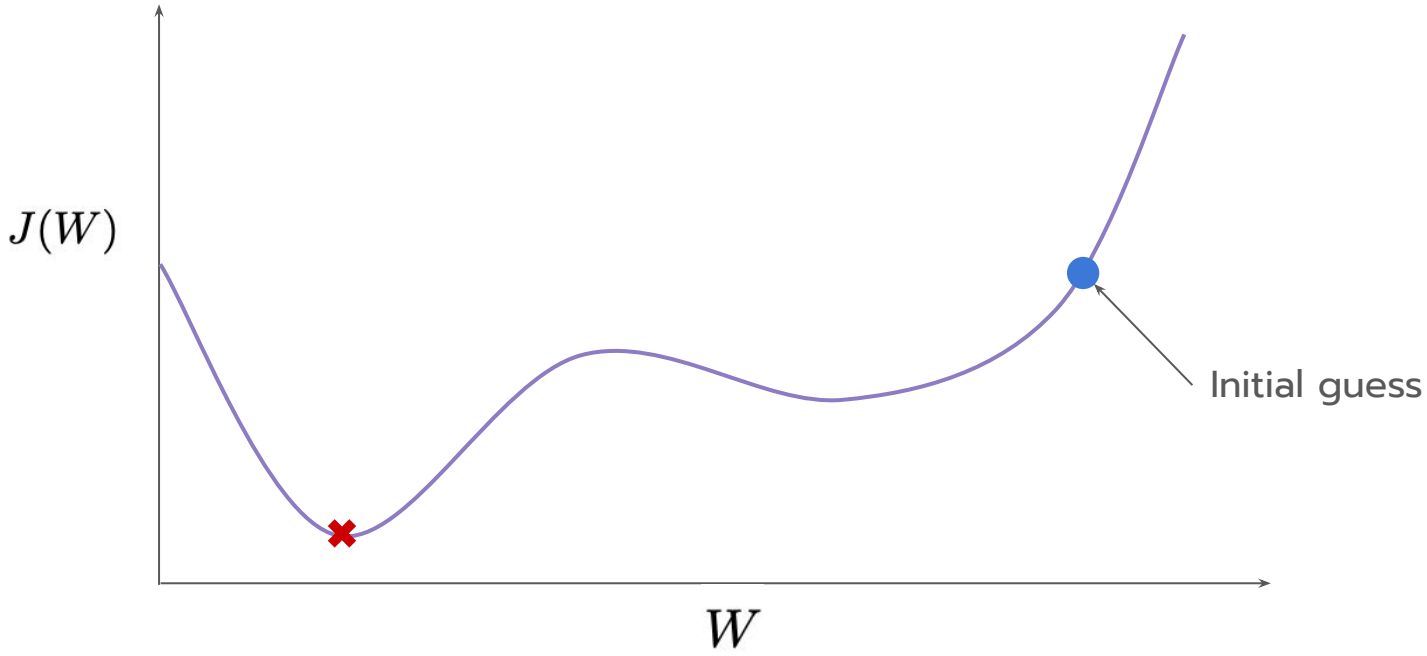


Slide adopted from: Alexander Amini, MIT 6.S191 Introduction to Deep Learning, MIT, http://introtodeeplearning.com/

Optimization

Setting the Learning rate

Large learning rate overshoot, become unstable and diverge

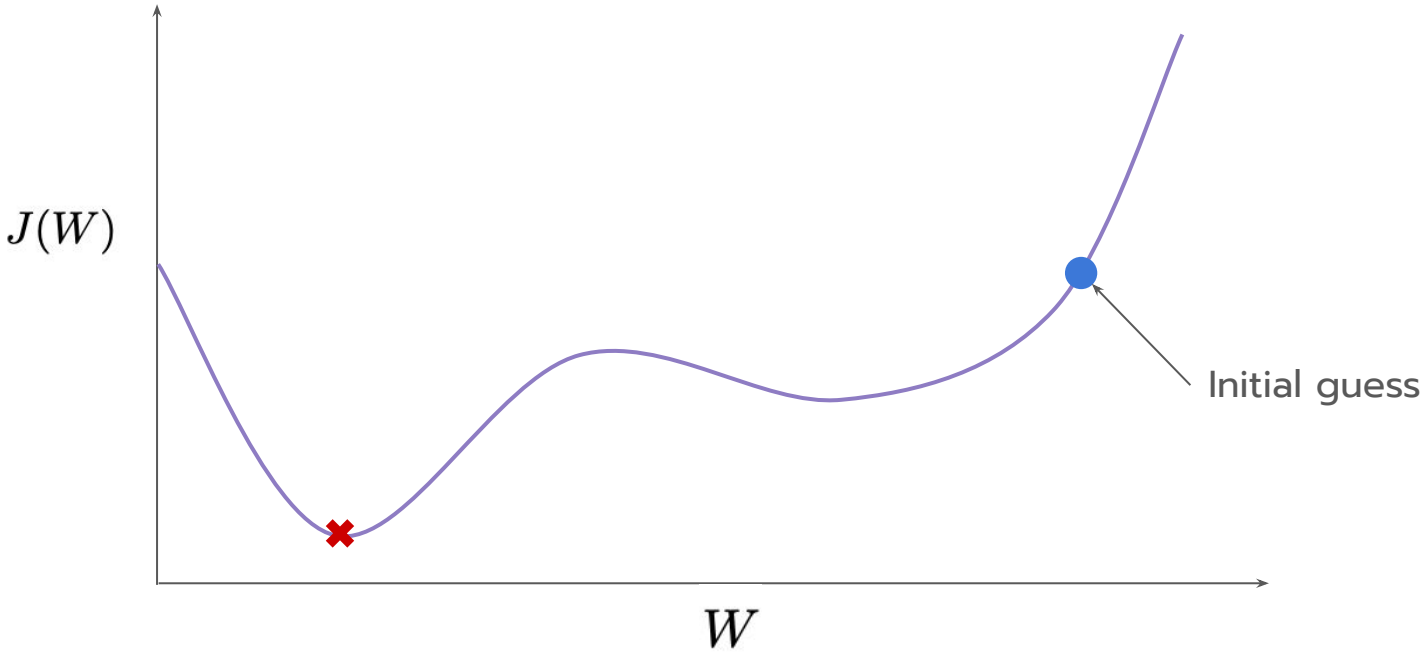


Slide adopted from: Alexander Amini, MIT 6.S191 Introduction to Deep Learning, MIT, http://introtodeeplearning.com/

Optimization

Setting the Learning rate

Stable learning rate converge <u>smoothly and avoid</u> local minima



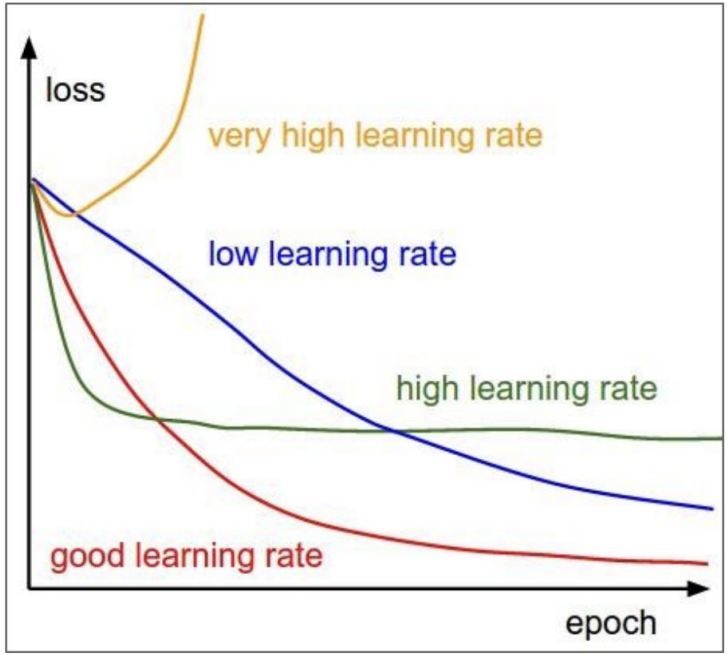
Slide adopted from: Alexander Amini, MIT 6.S191 Introduction to Deep Learning, MIT, http://introtodeeplearning.com/

Optimization

Setting the Learning rate: How to deal with this ?

$$W \longleftarrow W - \boxed{\eta} \frac{\partial J(W)}{\partial W}$$

Q: Which one of these learning rates is the best to use?



Slide adopted from: Fei Fei Li, CS231n: Deep Learning for Computer Vision, Stanford University, https://cs231n.stanford.edu/

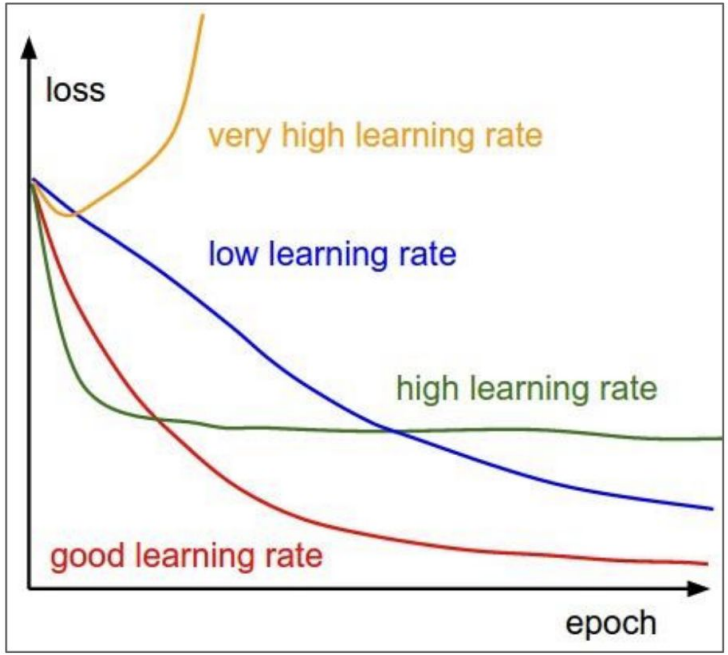
Optimization

Setting the Learning rate: How to deal with this ?

$$W \longleftarrow W - \boxed{\eta} \frac{\partial J(W)}{\partial W}$$

Q: Which one of these learning rates is the best to use?

A: In reality, all of these could be good learning rates.



Slide adopted from: Fei Fei Li, CS231n: Deep Learning for Computer Vision, Stanford University, https://cs231n.stanford.edu/

Optimization

Setting the Learning rate: How to deal with this ?

$$W \longleftarrow W - \boxed{\eta} \frac{\partial J(W)}{\partial W}$$

Adaptive Learning Rate:

- Learning rates are no longer fixed.
- Can be made larger or smaller depending on:
  - How large gradient is
  - How fast learning rate happening
  - Size of particular weights
  - etc...

Slide adopted from: Alexander Amini, MIT 6.S191 Introduction to Deep Learning, MIT, http://introtodeeplearning.com/

Optimization

Setting the Learning rate: Decay

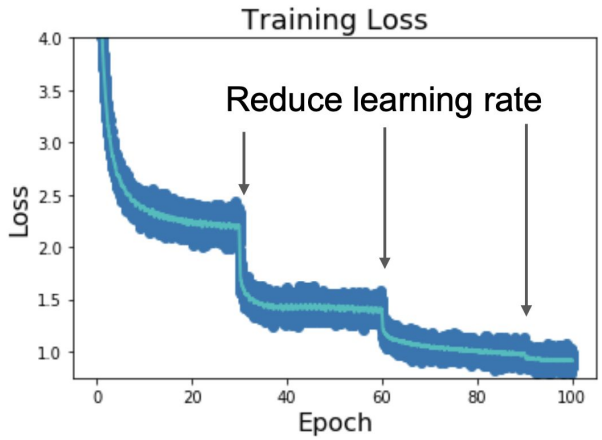
Step: Reduce learning rate at a few fixed points. E.g. for ResNets, multiply LR by 0.1 after epochs 30, 60, and 90.

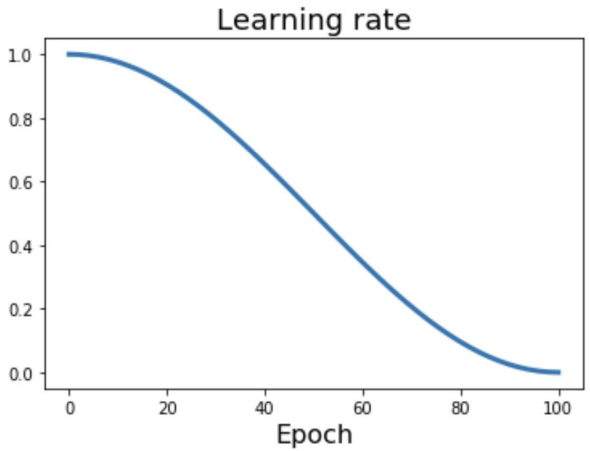
Cosine: $$\eta_t = \frac{1}{2}\eta_0(1 + cos(\frac{t\pi}{T})$$

η_t : Initial learning rate

η_0 : Learning rate at epoch t

T : Total number of epochs





Slide adopted from: Alexander Amini, MIT 6.S191 Introduction to Deep Learning, MIT, http://introtodeeplearning.com/

Optimization

Setting the Learning rate: Decay

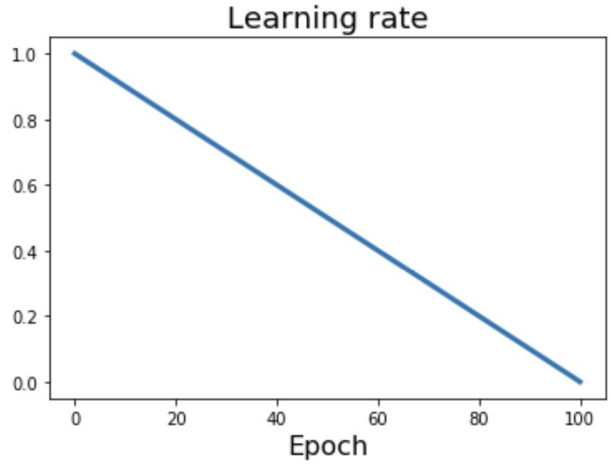
Linear: $\eta_t = \eta_0(1 - \frac{t}{T})$

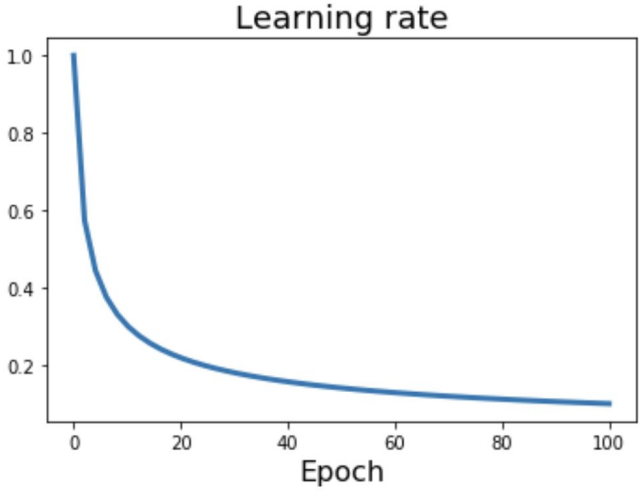
Inverse SQRT: $\eta_t = \frac{\eta_0}{\sqrt{T}}$

η_t : Initial learning rate

η_0 : Learning rate at epoch t

T : Total number of epochs





Slide adopted from: Alexander Amini, MIT 6.S191 Introduction to Deep Learning, MIT, http://introtodeeplearning.com/

Optimization

Gradient Descent Algorithms

- SGD
- Adam
- Adadelta
- Adagrad
- RMSProp

More Details https://www.ruder.io/optimizing-gradient-descent/

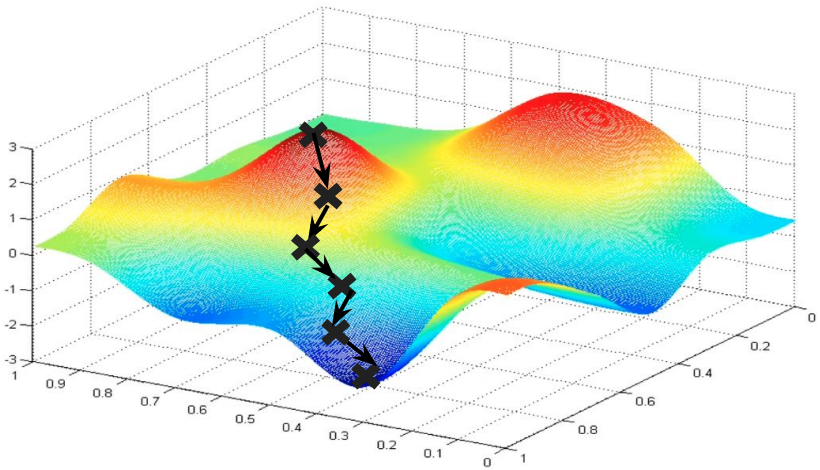
Slide adopted from: Alexander Amini, MIT 6.S191 Introduction to Deep Learning, MIT, http://introtodeeplearning.com/

Optimization

Stochastic Gradient Descent

Algorithm

1. Initialize weights randomly $\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
   a. Compute gradient $\frac{\partial J(W)}{\partial W}$
   b. Update weights $W \longleftarrow W - \eta \frac{\partial J(W)}{\partial W}$
3. Return weights



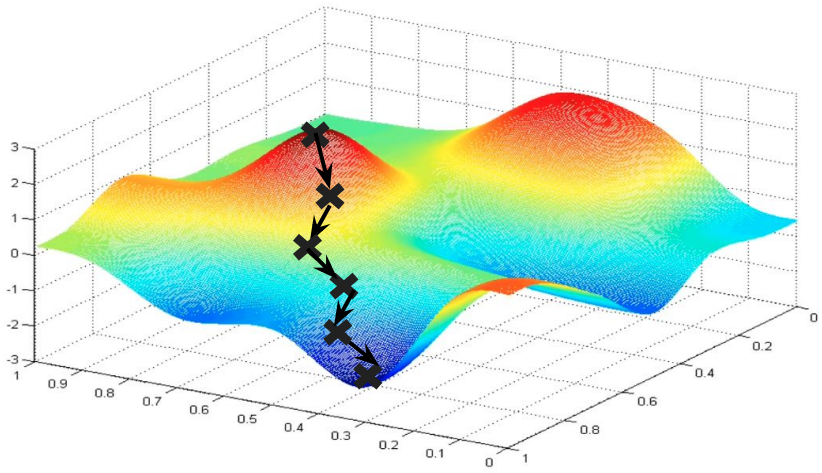
Slide adopted from: Alexander Amini, MIT 6.S191 Introduction to Deep Learning, MIT, http://introtodeeplearning.com/

Optimization

Stochastic Gradient Descent

Algorithm

1. Initialize weights randomly $\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
   a. Compute gradient $\boxed{\frac{\partial J(W)}{\partial W}}$
   b. Update weights $W \longleftarrow W - \eta \frac{\partial J(W)}{\partial W}$
3. Return weights



Can be very **computationally intensive** to compute.

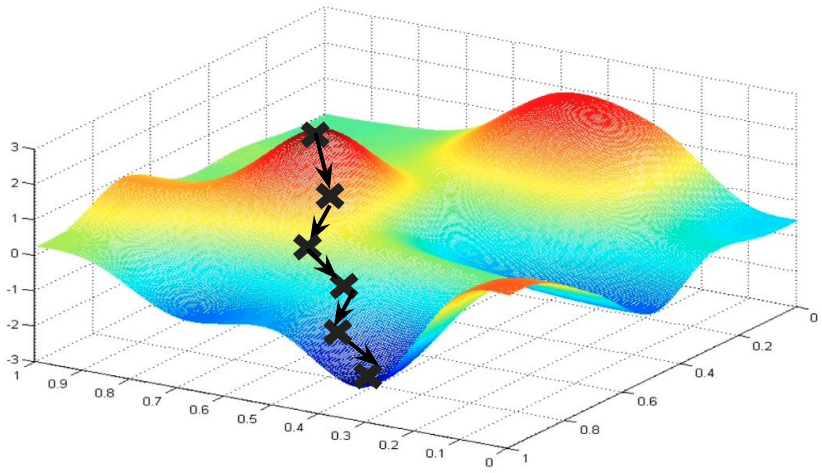
Slide adopted from: Alexander Amini, MIT 6.S191 Introduction to Deep Learning, MIT, http://introtodeeplearning.com/

Optimization

Stochastic Gradient Descent

Algorithm

1. Initialize weights randomly $\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
   a. Pick single data point i
   b. Compute gradient $\frac{\partial J_i(W)}{\partial W}$
   c. Update weights $W \longleftarrow W - \eta \frac{\partial J(W)}{\partial W}$
3. Return weights



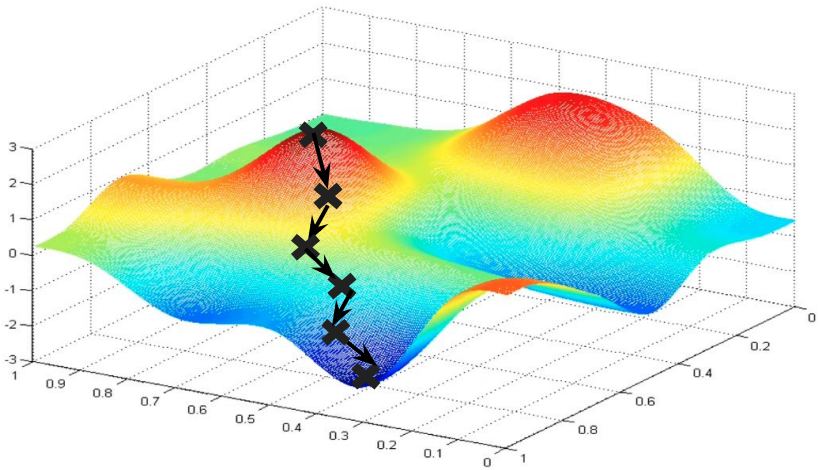
Slide adopted from: Alexander Amini, MIT 6.S191 Introduction to Deep Learning, MIT, http://introtodeeplearning.com/

Optimization

Stochastic Gradient Descent

Algorithm

1. Initialize weights randomly $\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
   a. Pick single data point i
   b. Compute gradient $\boxed{\frac{\partial J_i(W)}{\partial W}}$
   c. Update weights $W \longleftarrow W - \eta \frac{\partial J(W)}{\partial W}$
3. Return weights



Easy to compute, but **very noise** (stochastic).

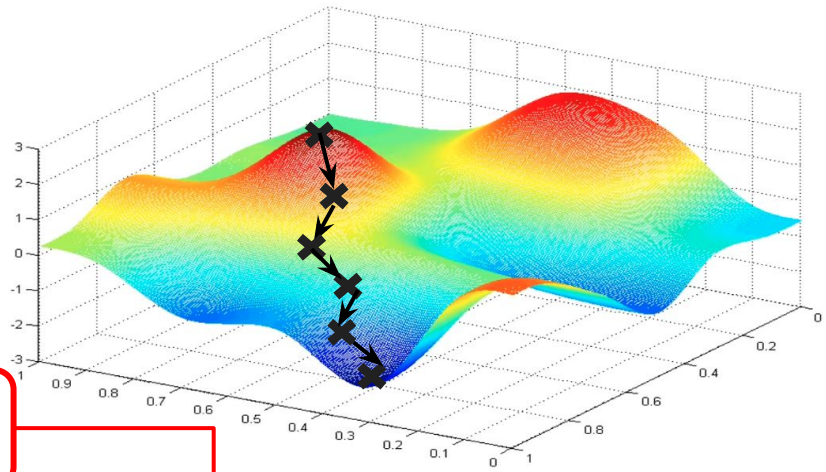
Slide adopted from: Alexander Amini, MIT 6.S191 Introduction to Deep Learning, MIT, http://introtodeeplearning.com/

Optimization

Stochastic Gradient Descent: mini-batches

Algorithm

1. Initialize weights randomly $\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
   a. Pick batch of ***B*** data points
   b. Compute gradient $\boxed{\frac{\partial J(W)}{\partial W} = \frac{1}{B}\sum_{k=1}^{B}\frac{\partial J_k(W)}{\partial W}}$
   c. Update weights $W \longleftarrow W - \eta\frac{\partial J(W)}{\partial W}$
3. Return weights



Fast to compute and much **better estimate** of the true gradient.

Slide adopted from: Alexander Amini, MIT 6.S191 Introduction to Deep Learning, MIT, http://introtodeeplearning.com/

Optimization

Stochastic Gradient Descent: mini-batches

Mini batches while Training:

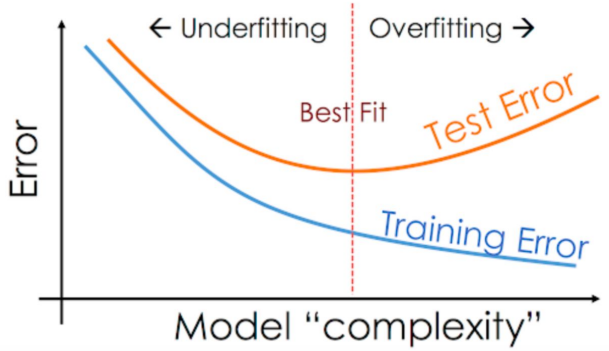
More Accurate Estimation of Gradient
Smooth convergence
Allows for larger learning rates

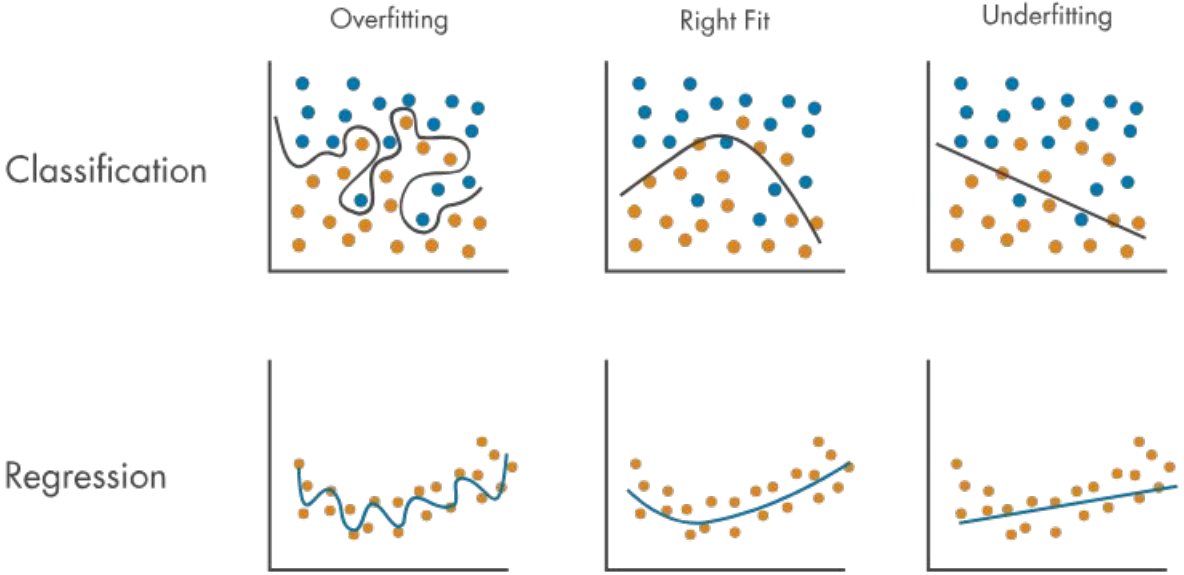
Lead to Fast training
Can parallelize computation + achieve significant speed increases in GPU's

Slide adopted from: Alexander Amini, MIT 6.S191 Introduction to Deep Learning, MIT, http://introtodeeplearning.com/

Optimization

The problem of Overfitting





Too complex, extra parameters, does not generalize well

- Low **Training** loss, but high on **Testing**
- High **Training** accuracy, low on **testing**

Ideal fit

- Low loss
- High accuracy

Model does not have capacity to fully learn the data

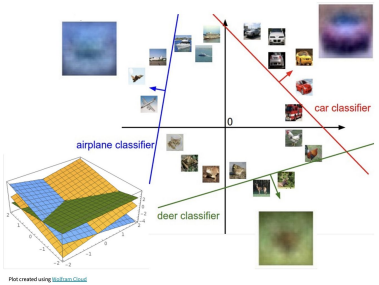
- High loss
- Low accuracy

Image from: https://www.mathworks.com/discovery/overfitting.html

Summary

Use **Linear Model** for image classification problems

$$s = f(x; W) = Wx$$



Use **Loss Function** to express preferences over different choices of weights

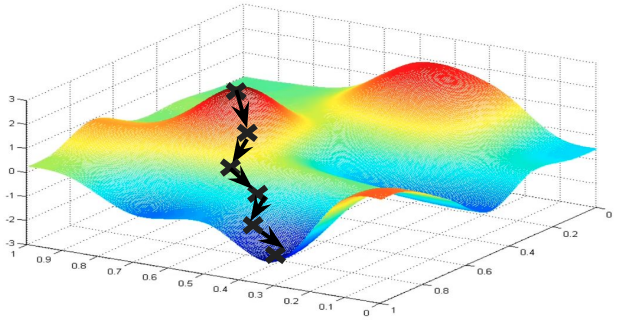
$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$ Softmax

SVM

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i + R(W)$$ Full loss

Use **Stochastic Gradient Descent** to minimize our loss functions and train the model

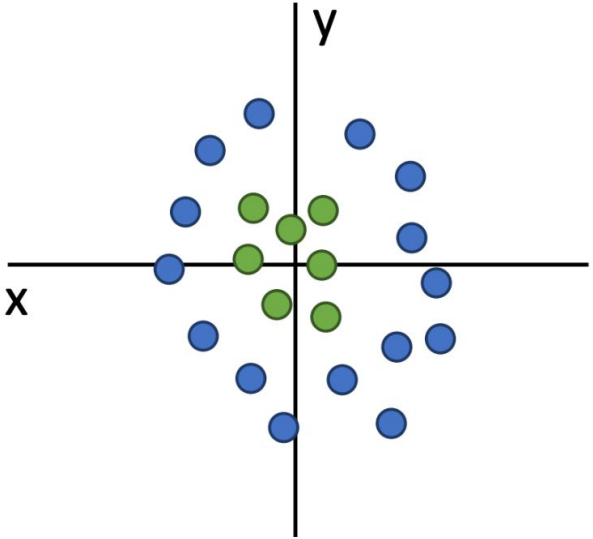


Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Linear Classifier

Problem:

Geometric Viewpoint



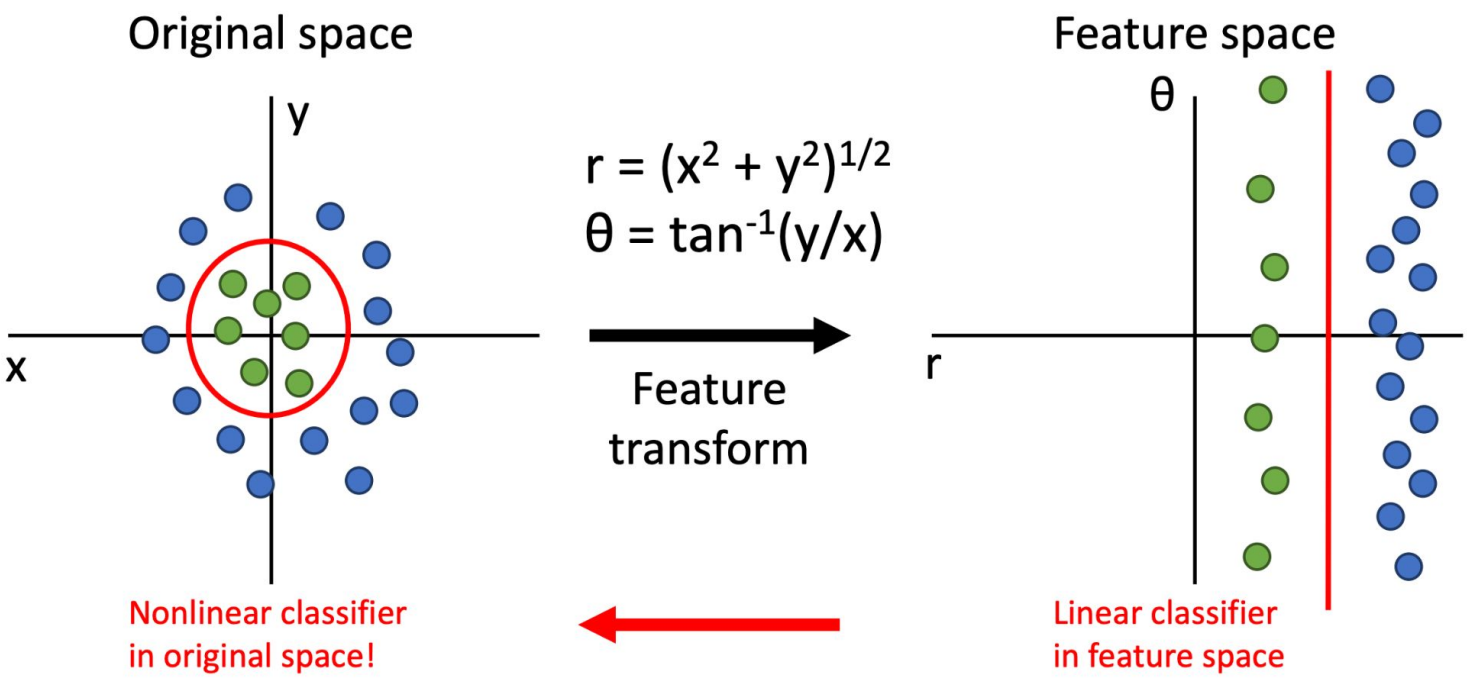
Visual Viewpoint

One template per class:
Can't recognize different
modes of a class

Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Linear Classifier

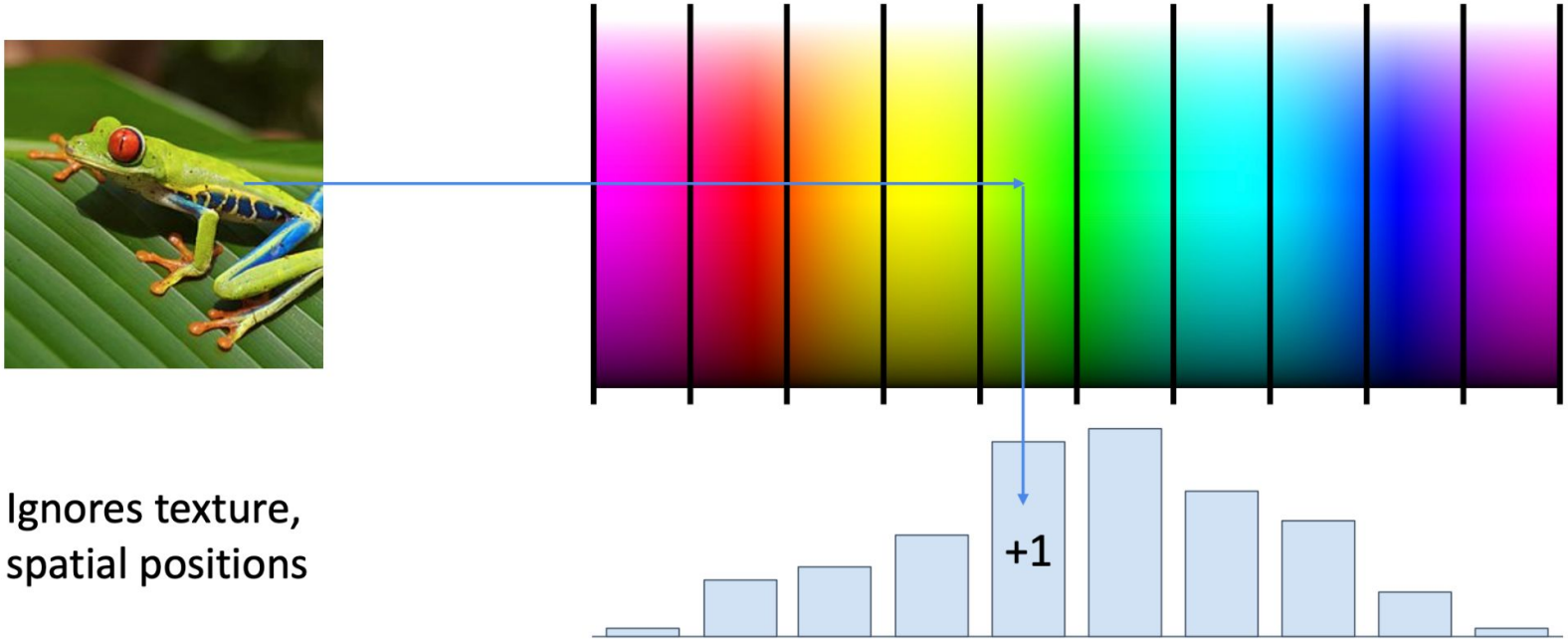
Solution: Feature Transforms



Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Linear Classifier

Image Features: Color Histogram

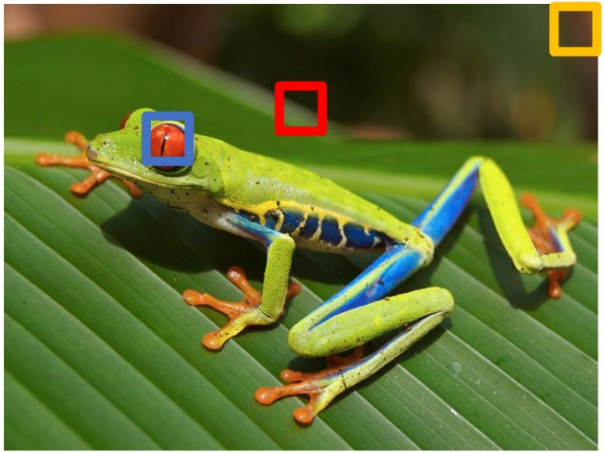


Ignores texture,
spatial positions

Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Linear Classifier

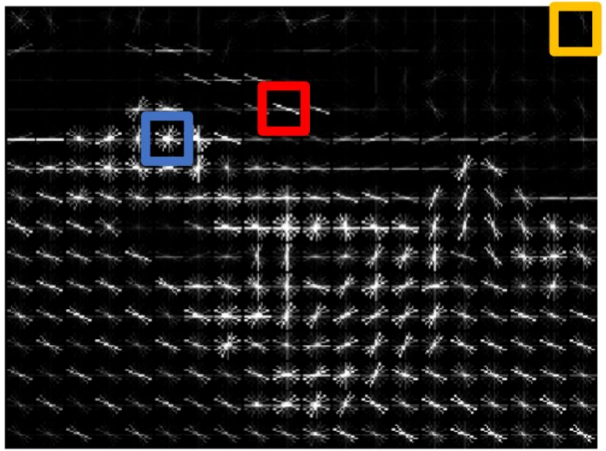
Image Features: Histogram of Oriented Gradients (HoG)



Weak edges

Strong diagonal edges

Edges in all directions



1. Compute edge direction / strength at each pixel
2. Divide image into 8x8 regions
3. Within each region compute a histogram of edge directions weighted by edge strength

Captures texture and position, robust to small image changes

Example: 320x240 image gets divided into 40x30 bins; 8 directions per bin; feature vector has 30*40*9 = 10,800 numbers

Lowe, "Object recognition from local scale-invariant features", ICCV 1999
Dalal and Triggs, "Histograms of oriented gradients for human detection," CVPR 2005

Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

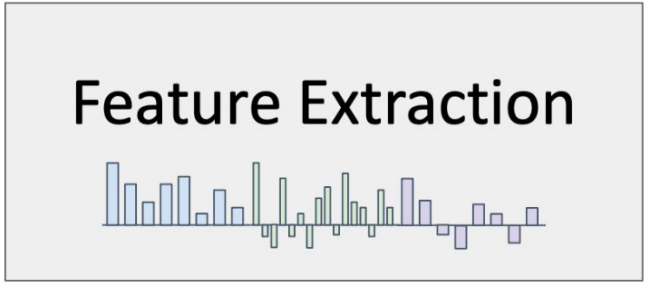
Linear Classifier

Image Features

Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Linear Classifier

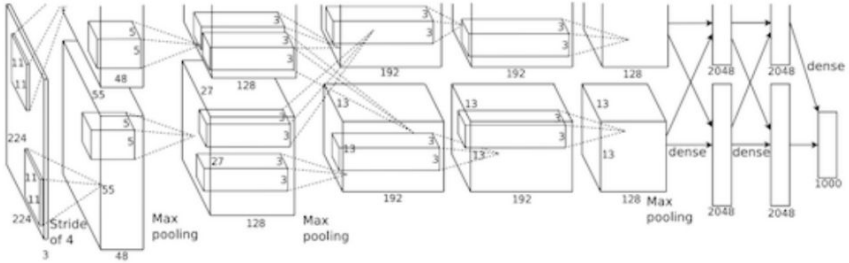
Image Features



f

10 numbers giving scores for classes

training



Krizhevsky, Sutskever, and Hinton, "Imagenet classification with deep convolutional neural networks", NIPS 2012. Figure copyright Krizhevsky, Sutskever, and Hinton, 2012. Reproduced with permission.

10 numbers giving scores for classes

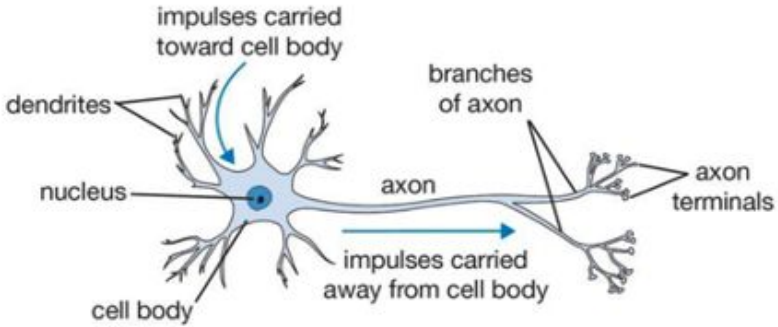
training

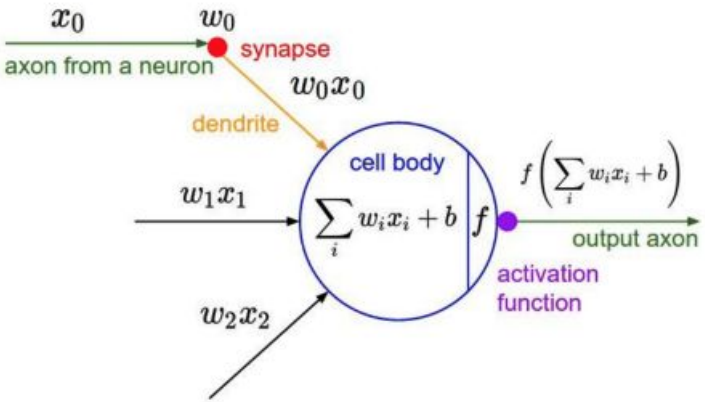
Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Neural Network

Neural Network

Biological vs Artificial Neuron

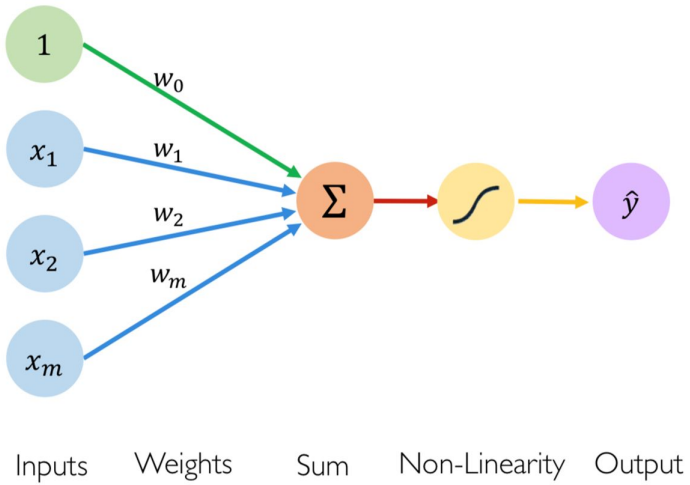




Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Neural Network

Perceptron: Forward Propagation



Linear combination of inputs

Output

$$\hat{y} = g(w_0 + \sum_{i=1}^{m} x_i w_i)$$

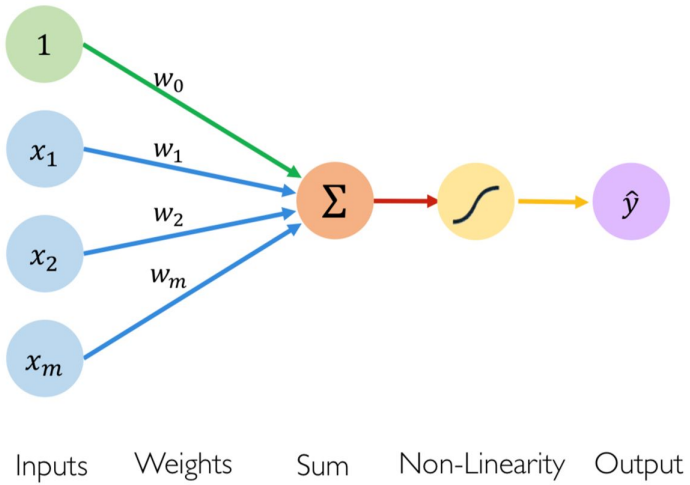
Non-linear activation function

Bias

Slide adopted from: Alexander Amini, MIT 6.S191 Introduction to Deep Learning, MIT, http://introtodeeplearning.com/

Neural Network

Perceptron: Forward Propagation



$$\hat{y} = g(w_0 + \sum_{i=1}^{m} x_i w_i)$$

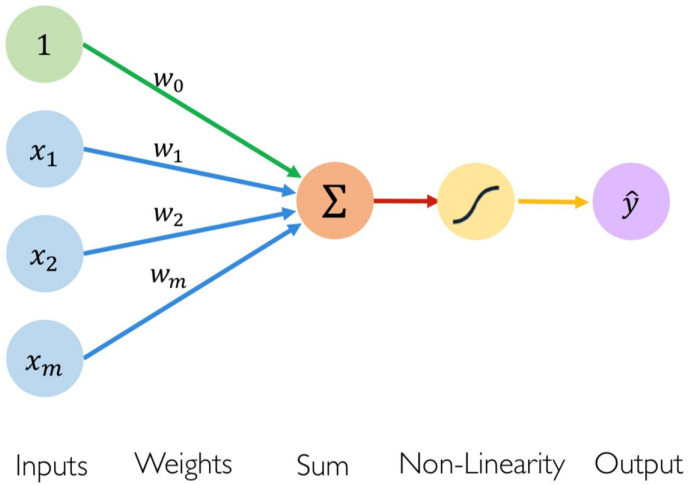
$$\hat{y} = g(w_0 + X^T W)$$

where: $X = \begin{bmatrix} x_1 \\ \vdots \\ w_m \end{bmatrix}$ and $W = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$

Slide adopted from: Alexander Amini, MIT 6.S191 Introduction to Deep Learning, MIT, http://introtodeeplearning.com/

Neural Network

Perceptron: Activation Function



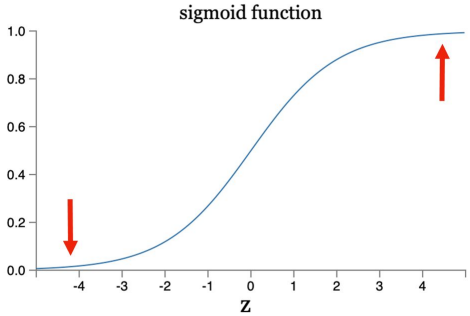
Activation Function

$$\hat{y} = \boxed{g}(w_0 + X^T W)$$

Example: Sigmoid function

$$\sigma(z) \equiv \frac{1}{1+e^{-z}}.$$

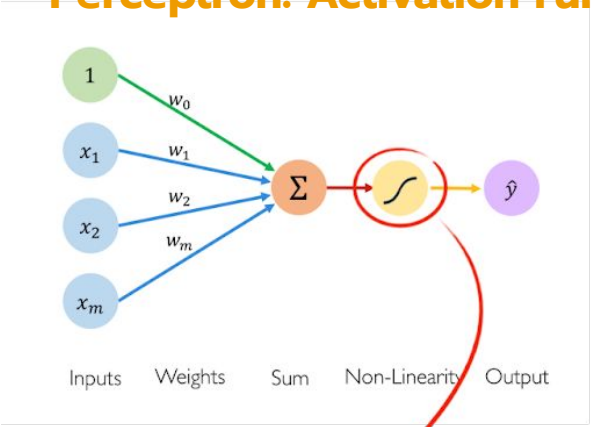
$$z = w \cdot x + b$$

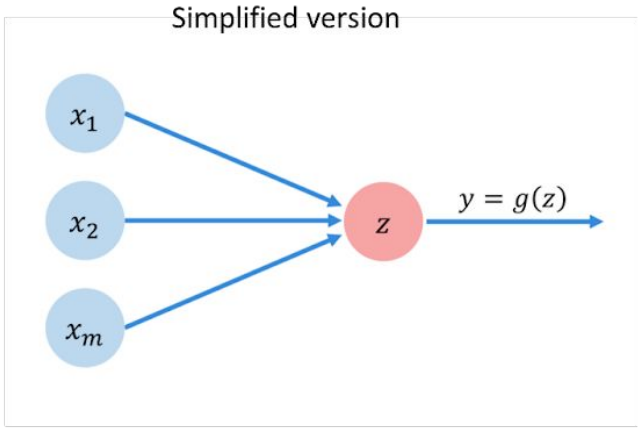


Slide adopted from: Alexander Amini, MIT 6.S191 Introduction to Deep Learning, MIT, http://introtodeeplearning.com/

Neural Network

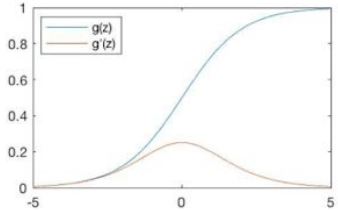
Perceptron: Activation Function



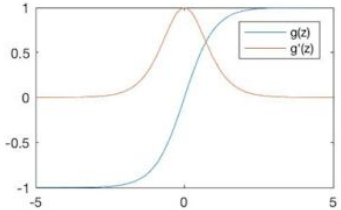


Activation functions

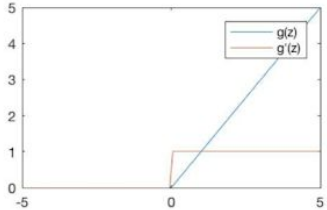
- ReLU
- Tanh
- LeakyReLU



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



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



$$g(z) = \max(0, z)$$

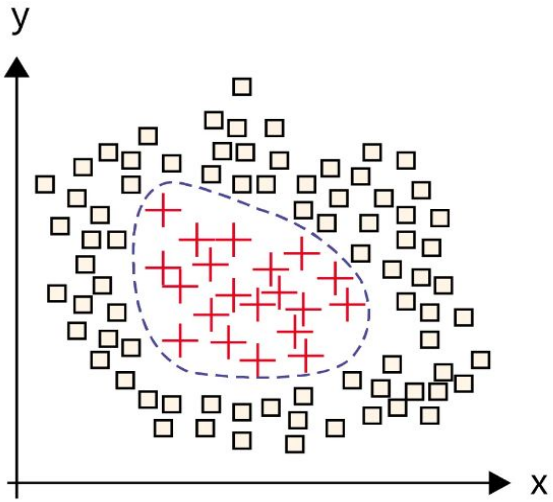
Slide adopted from: Alexander Amini, MIT 6.S191 Introduction to Deep Learning, MIT, http://introtodeeplearning.com/

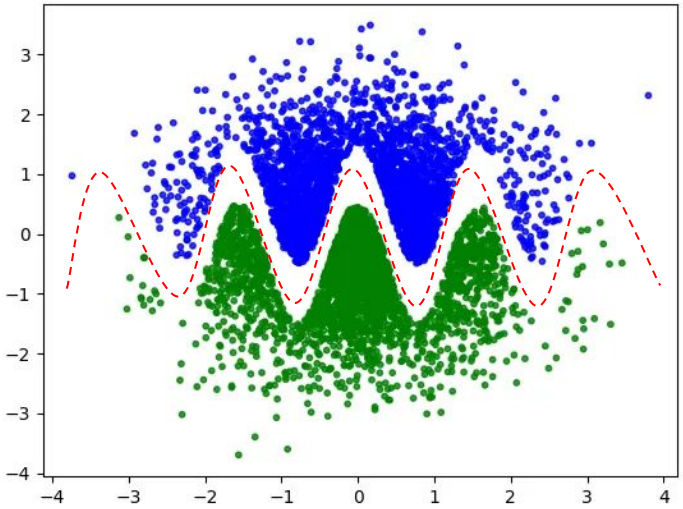
Neural Network

Perceptron: Activation Function

The purpose of activation functions is to "**introduce non-linearities**" into the network.

Non-linearities allows us to approximate arbitrarily complex functions.





Slide adopted from: Alexander Amini, MIT 6.S191 Introduction to Deep Learning, MIT, http://introtodeeplearning.com/

Neural Network

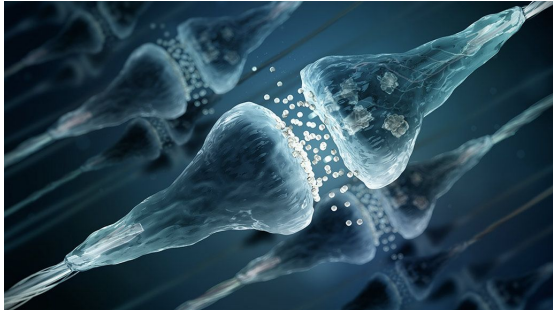
Perceptron: Activation Function

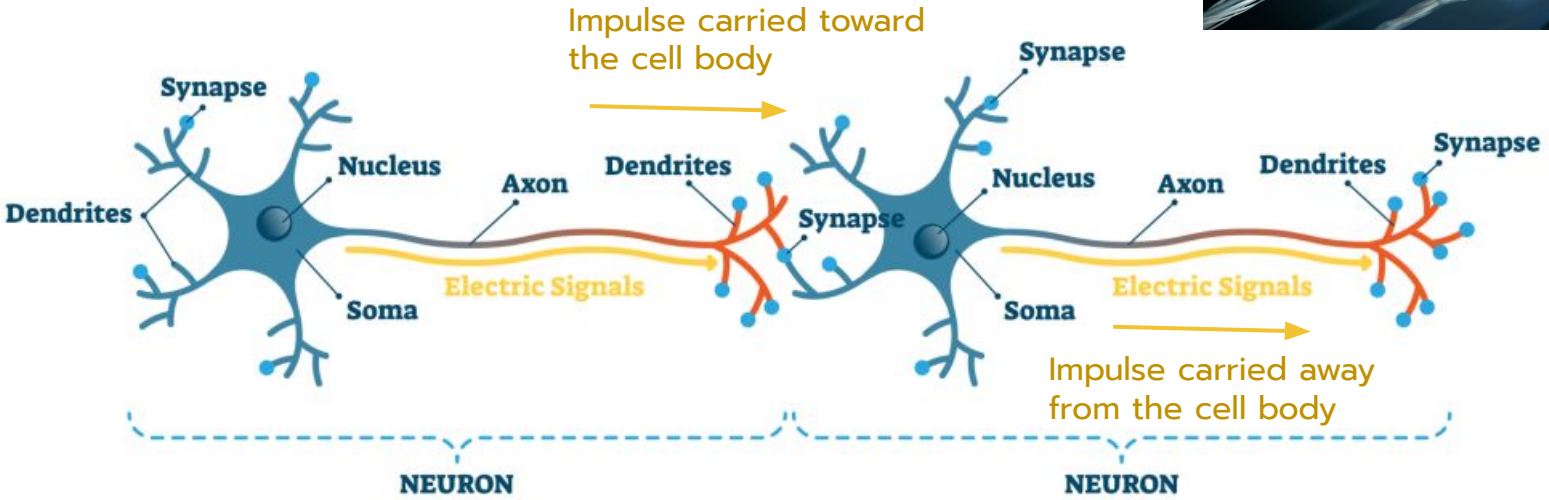
Properties of Activation Functions

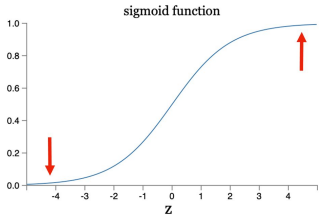
- **Differentiable**: We have to differentiate all the neurons during the **backpropagation** to optimize the loss function. That's why all the activation functions are differentiable.
- **Computational Expense**: Activation functions are applied after every layer and need to be calculated millions of times in deep networks. Hence, they should be computationally **inexpensive to calculate**.
- **Monotonicity:** Most of the activation functions are monotonic in nature i.e it is either **always decreasing** or **always increasing** which can simplify optimization.

Neural Network

Perceptron: Activation Function







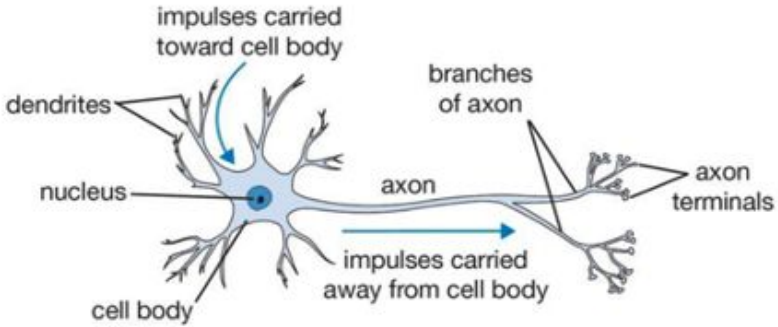
Firing rate is a nonlinear function of inputs

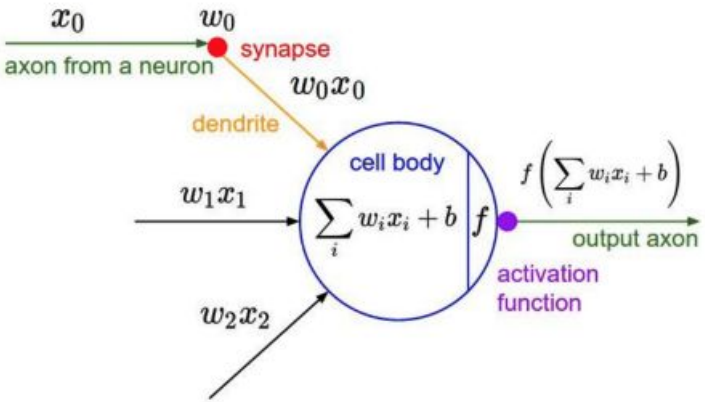
Image from: https://www.simplypsychology.org/neuron.html

Cherdsak Kingkan

Neural Network

Biological vs Artificial Neuron



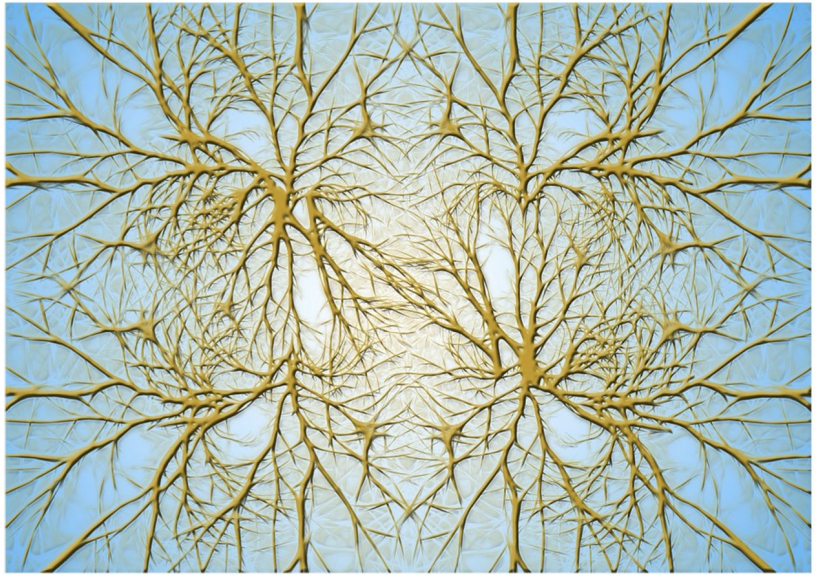


Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

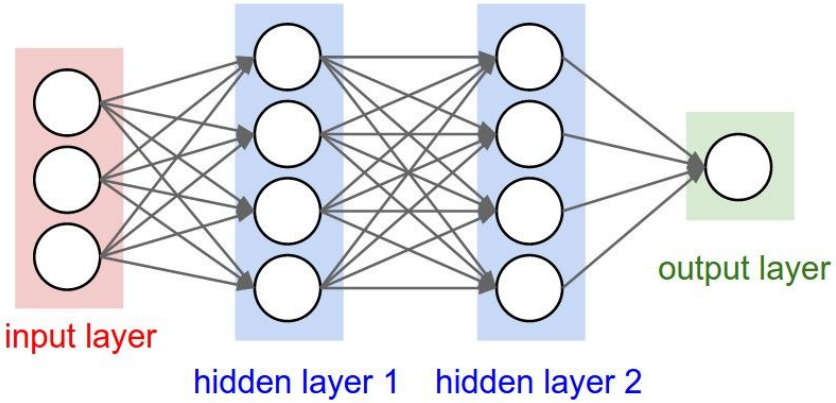
Neural Network

Architectures

Biological Neurons:
Complex connectivity patterns.



Neurons in a neural network:
Organized into regular layers for computational efficiency

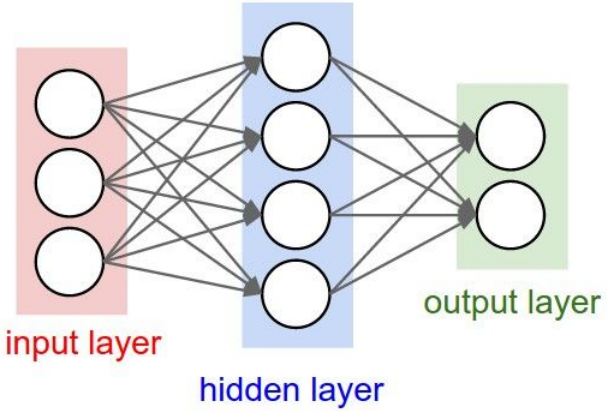


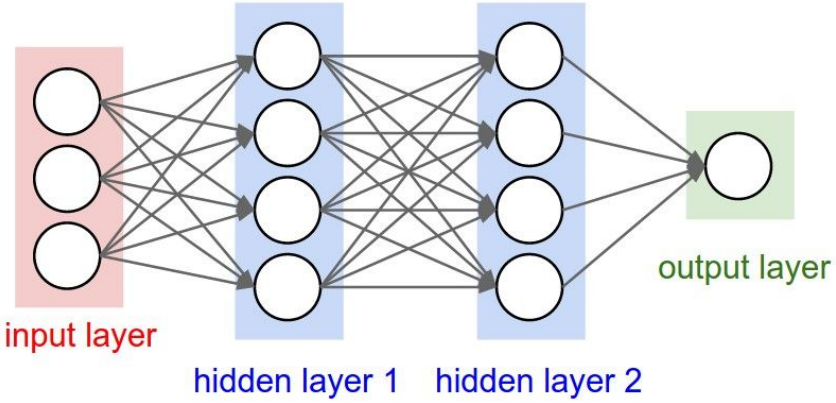
Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Neural Network

Architectures

Neural Network is a collection of neurons.
Multiple layer network is called "**Multi-Layer Perceptrons (MLP)**" or sometime "**Fully-connected network**".



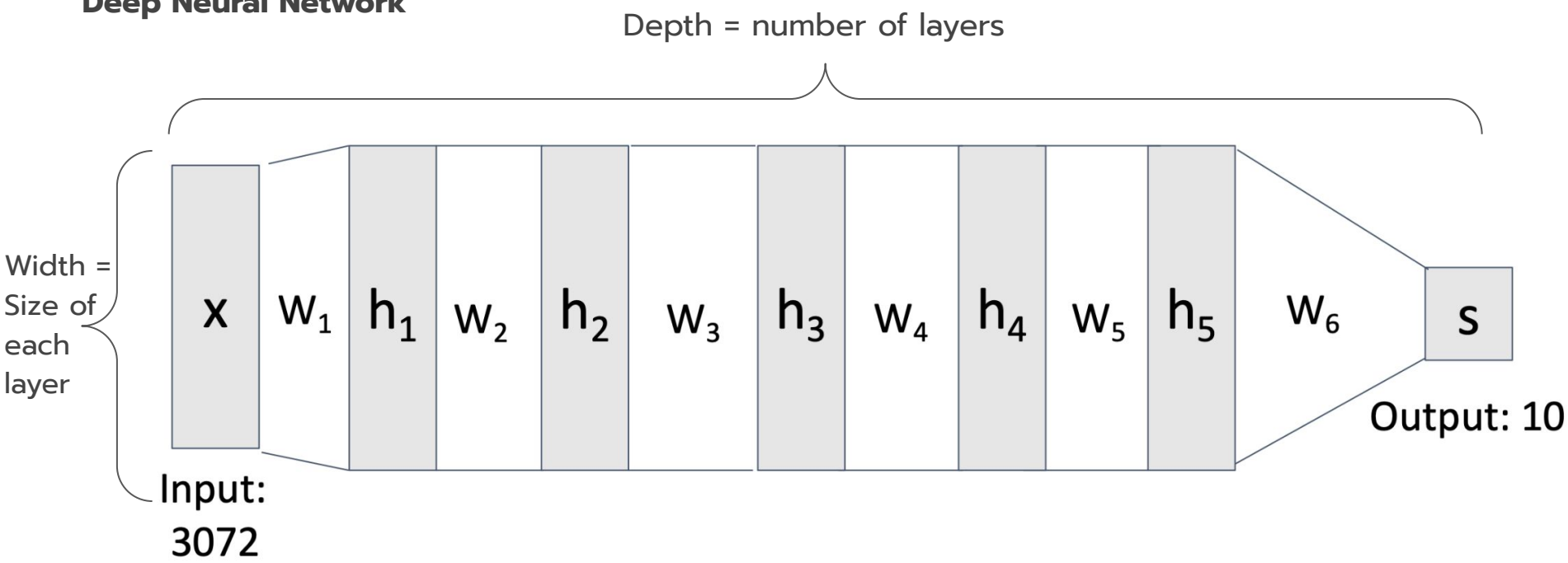


Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Neural Network

Architectures

Deep Neural Network



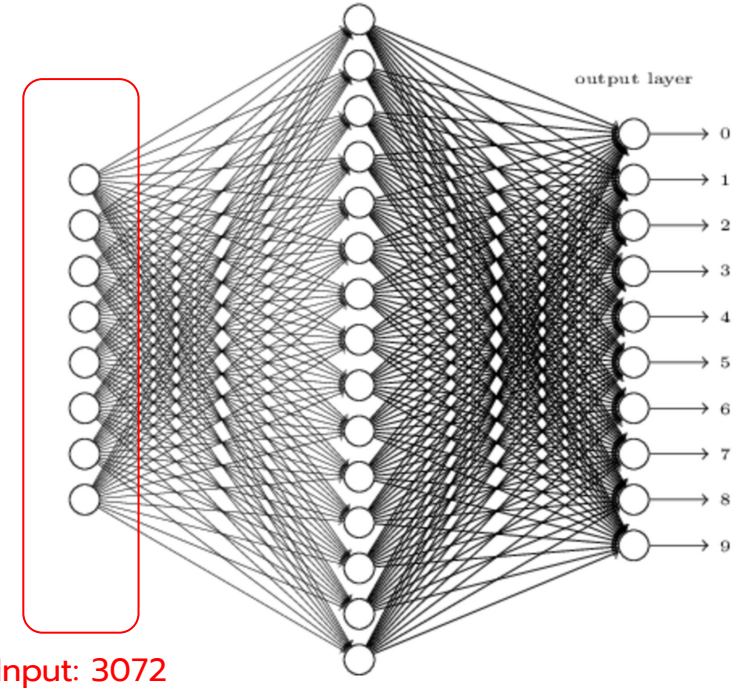
Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Neural Network

Architectures

Image

Array of 32x32x3 numbers
(3072 numbers total)



Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

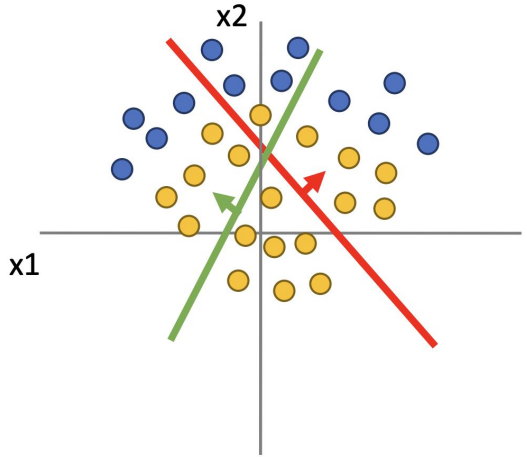
Neural Network

Why NN is a powerful function approximation

Space Warping

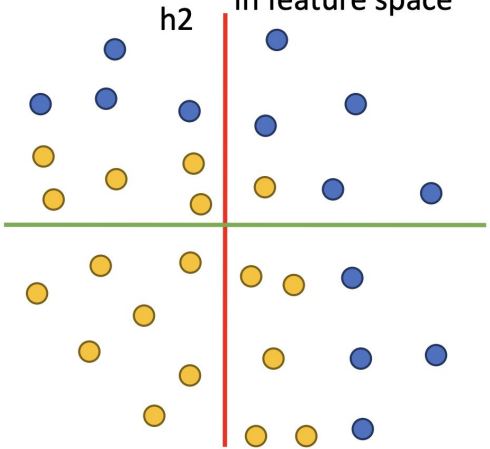
Consider a linear transform: $h = Wx$ Where x, h are both 2-dimensional

Points not linearly separable in original space



Feature transform:
h = Wx

Not linearly separable in feature space



Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Neural Network

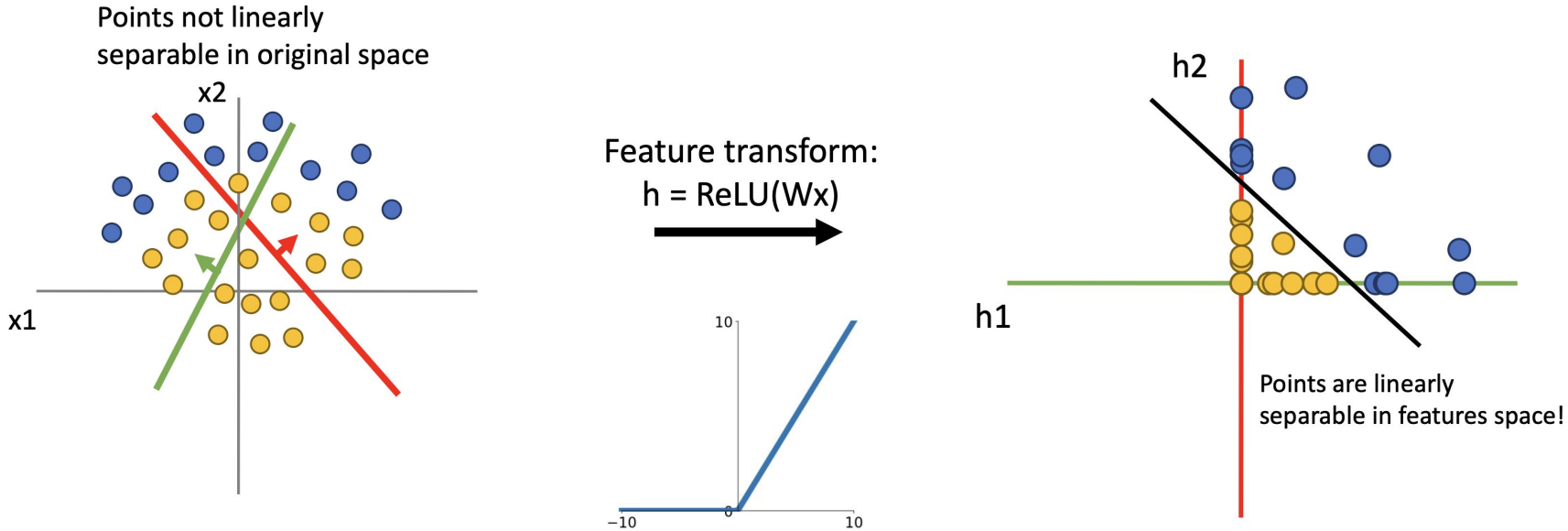
Why NN is a powerful function approximation

Space Warping

Consider a neural net hidden layer:

$h = \mathbf{ReLU}(Wx) = max(0, Wx)$

Where x, h are both 2-dimensional



Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Neural Network

Why NN is a powerful function approximation

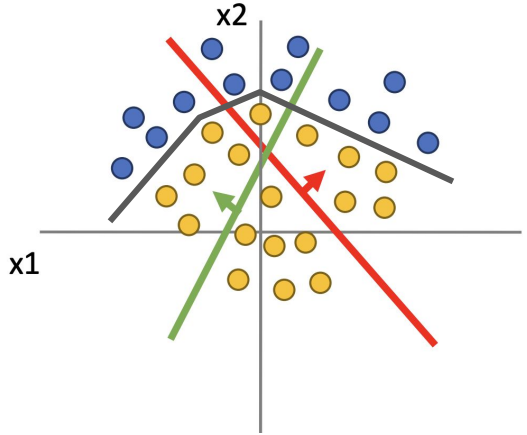
Space Warping

Consider a neural net hidden layer:

$h = \mathbf{ReLU}(Wx) = max(0, Wx)$

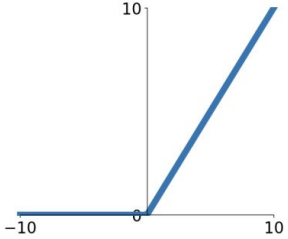
Where x, h are both 2-dimensional

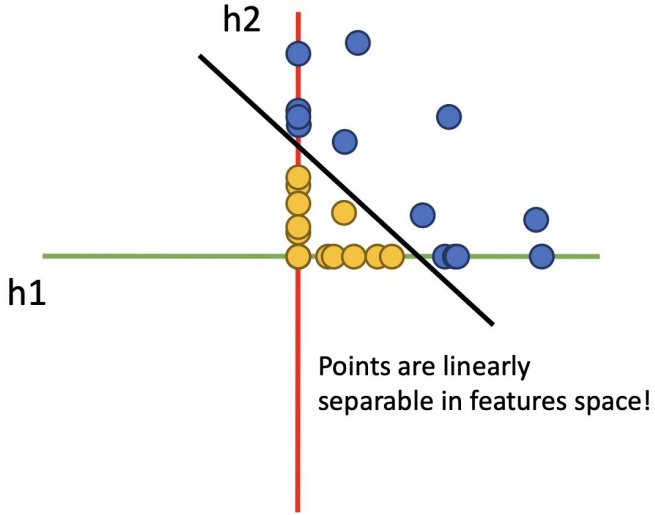
Points not linearly separable in original space



Linear classifier in feature space gives nonlinear classifier in original space

Feature transform:
h = ReLU(Wx)

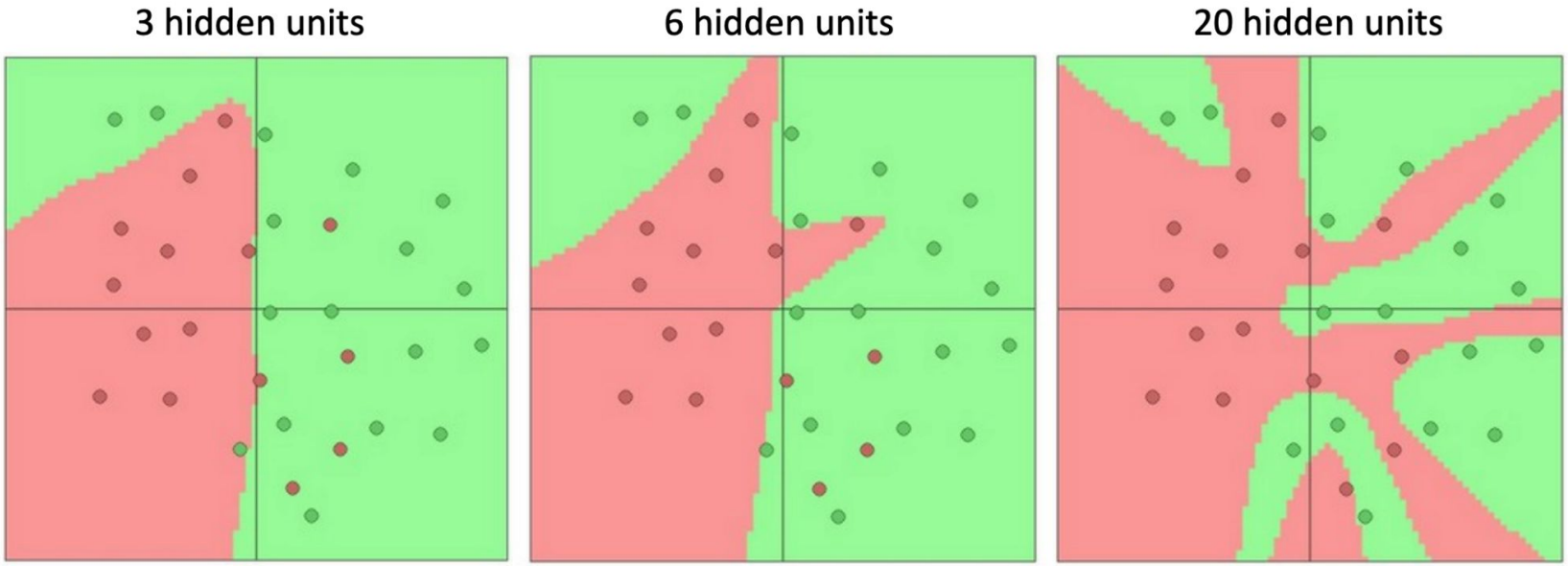




Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Neural Network

Setting the number of layers and their sizes



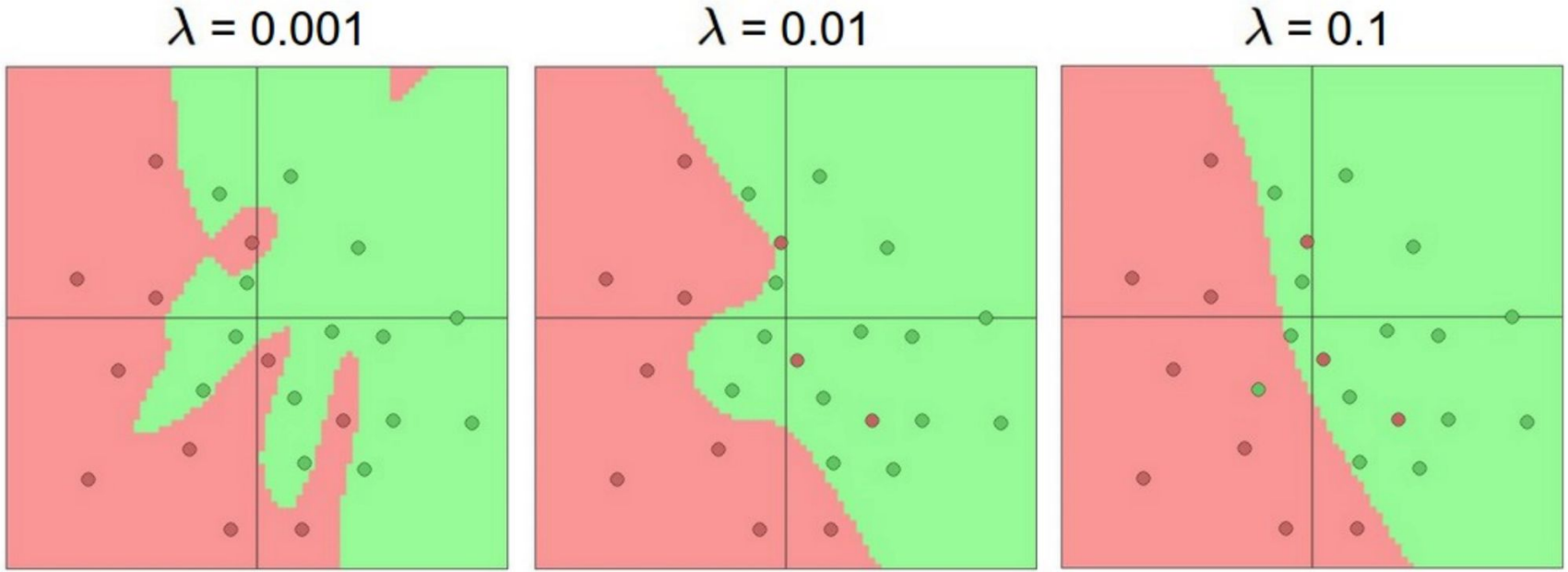
More hidden units = more capacity = more complex decision boundary

Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Neural Network

Setting the number of layers and their sizes

Don't regularize with size; instead **use stronger L2**



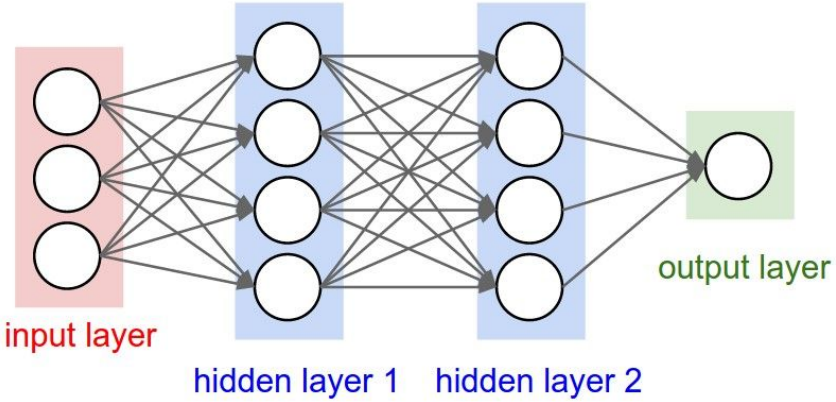
Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Neural Network

Perceptron: Activation Function

Q: What happens if we build a neural network with no activation function *g*?

$$\hat{y} = g(w_0 + \sum_{i=1}^{m} x_i w_i)$$



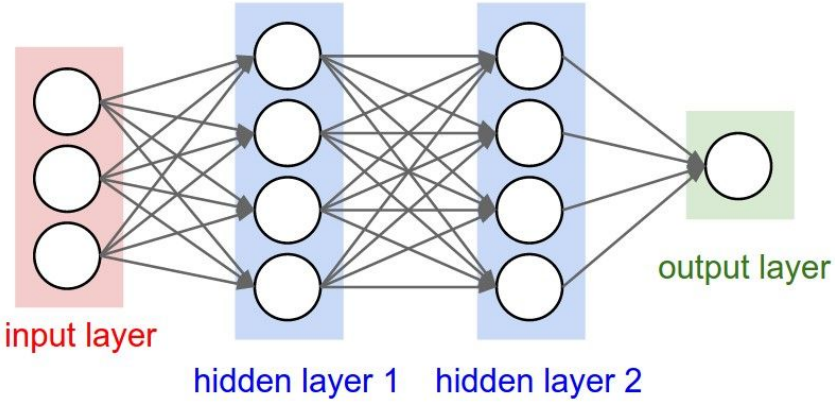
Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Neural Network

Perceptron: Activation Function

Q: What happens if we build a neural network with no activation function *g*?

$$\hat{y} = g\left(\boxed{w_0 + \sum_{i=1}^{m} x_i w_i}\right)$$



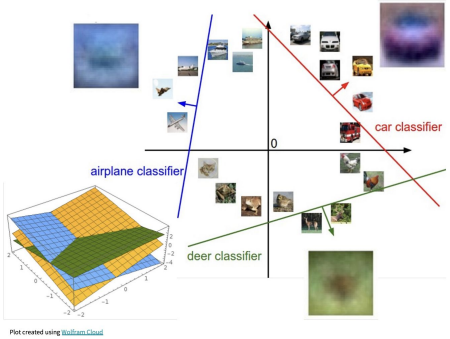
A: We end up with a linear classifier!

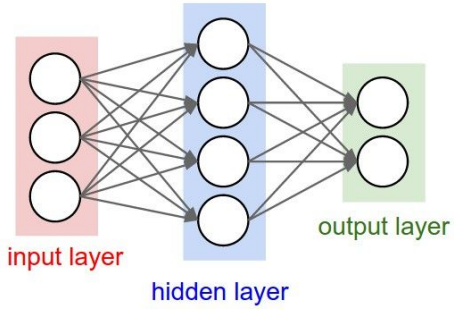
Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Neural Network

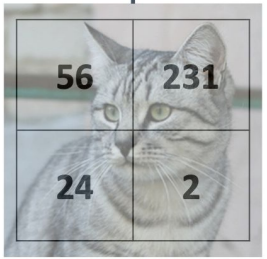
Problems of Linear classifiers and MLP

$$f(x,W) = Wx$$





Stretch pixels into column



Input image
(2, 2)

Problems:

- **Spatial structure** of images is ignore!
- Many many parameters

56
231
24
2

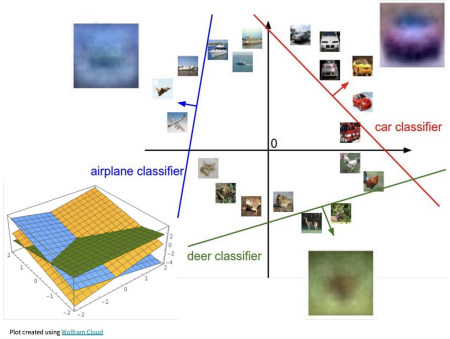
(4,)

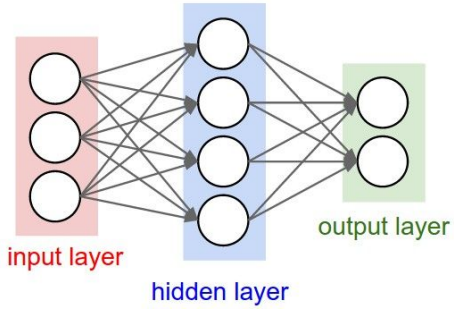
Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Neural Network

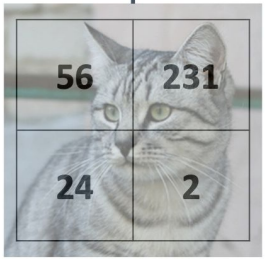
Problems of Linear classifiers and MLP

$$f(x,W) = Wx$$





Stretch pixels into column



Input image
(2, 2)

Problems:

- **Spatial structure** of images is ignore!
- Many many parameters

Solution: Define new operators that can operate on spatial data.

56
231
24
2

(4,)

Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

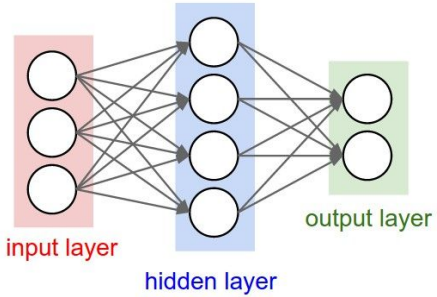
Convolutional

Neural Network

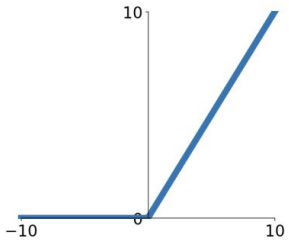
Convolutional Neural Network

Components of a Fully-Connected Network

Fully-Connected Layer



Activation Function

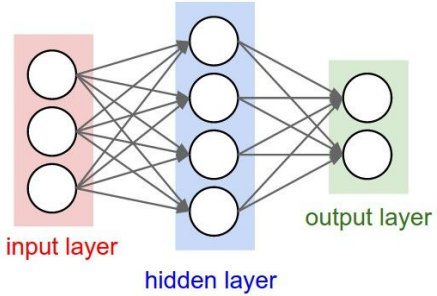


Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

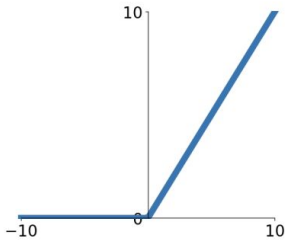
Convolutional Neural Network

Components of a Convolutional Network

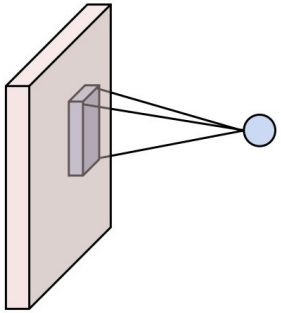
Fully-Connected Layer



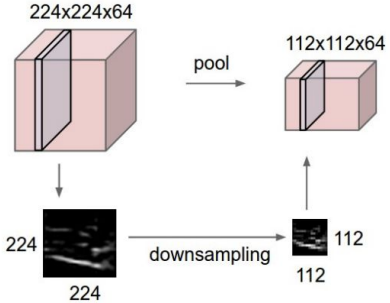
Activation Function



Convolution Layers



Pooling Layers



Normalization

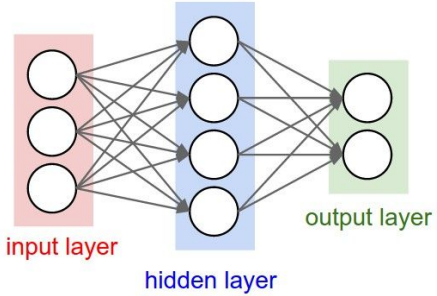
$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

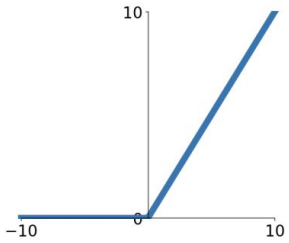
Convolutional Neural Network

Components of a Convolutional Network

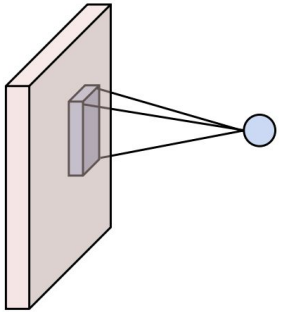
Fully-Connected Layer



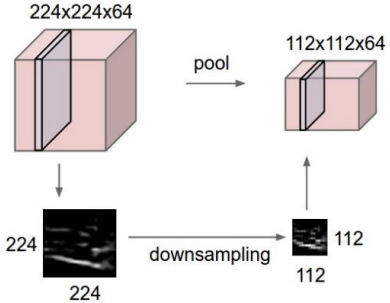
Activation Function



Convolution Layers



Pooling Layers



Normalization

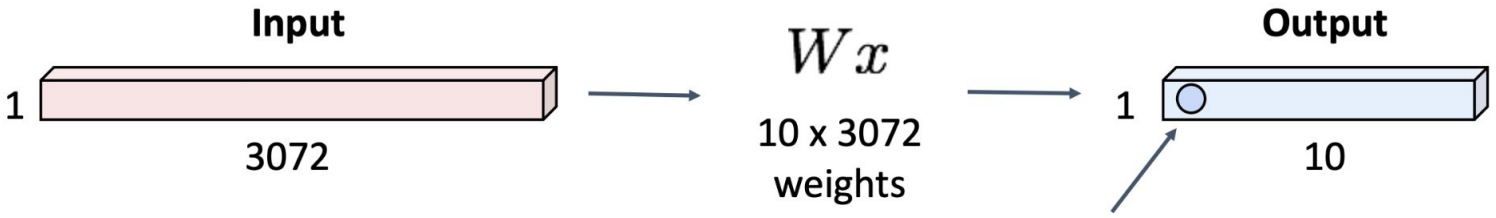
$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Convolutional Neural Network

Fully-Connected Layer

32 x 32 x 3 image -> stretch to 3072 x 1



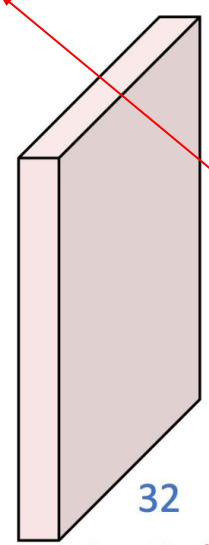
1 number:
the result of taking a dot product between a row of W and the input (a 3072-dimensional dot product)

Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Convolutional Neural Network

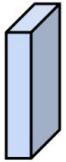
Convolution Layer

3x32x32 image :Preserve spatial structure



Filters always extend the full depth of input volume

3x5x5 filter



32 height

32 width

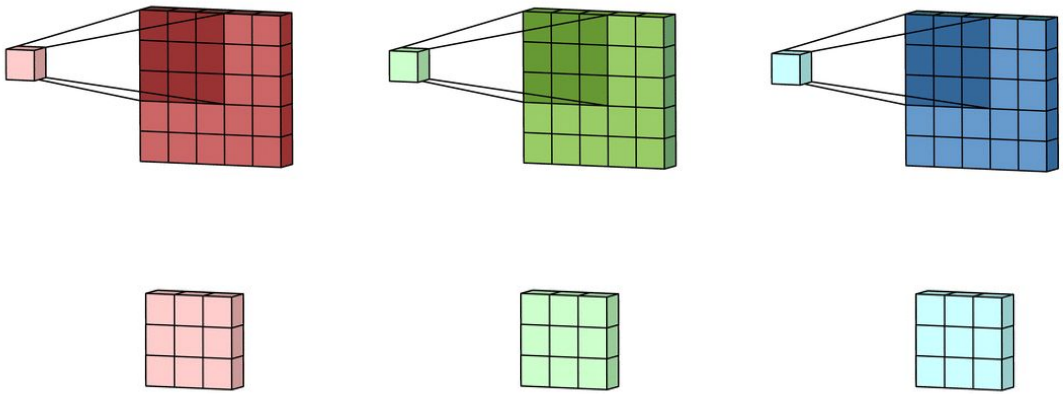
3 depth / channels

Convolve the filter with the image i.e. "slide over the image spatially, computing dot products"

Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

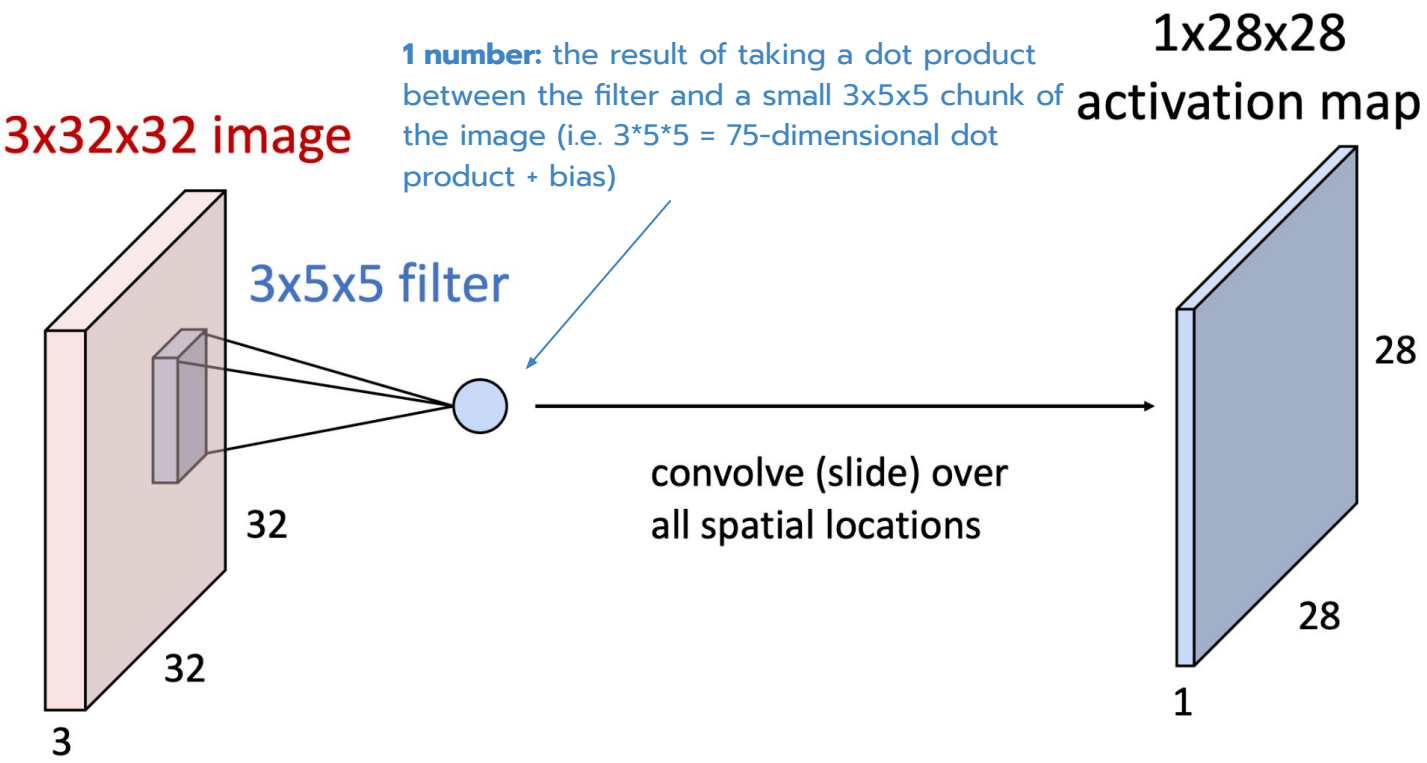
Convolutional Neural Network

Convolution Layer



Convolutional Neural Network

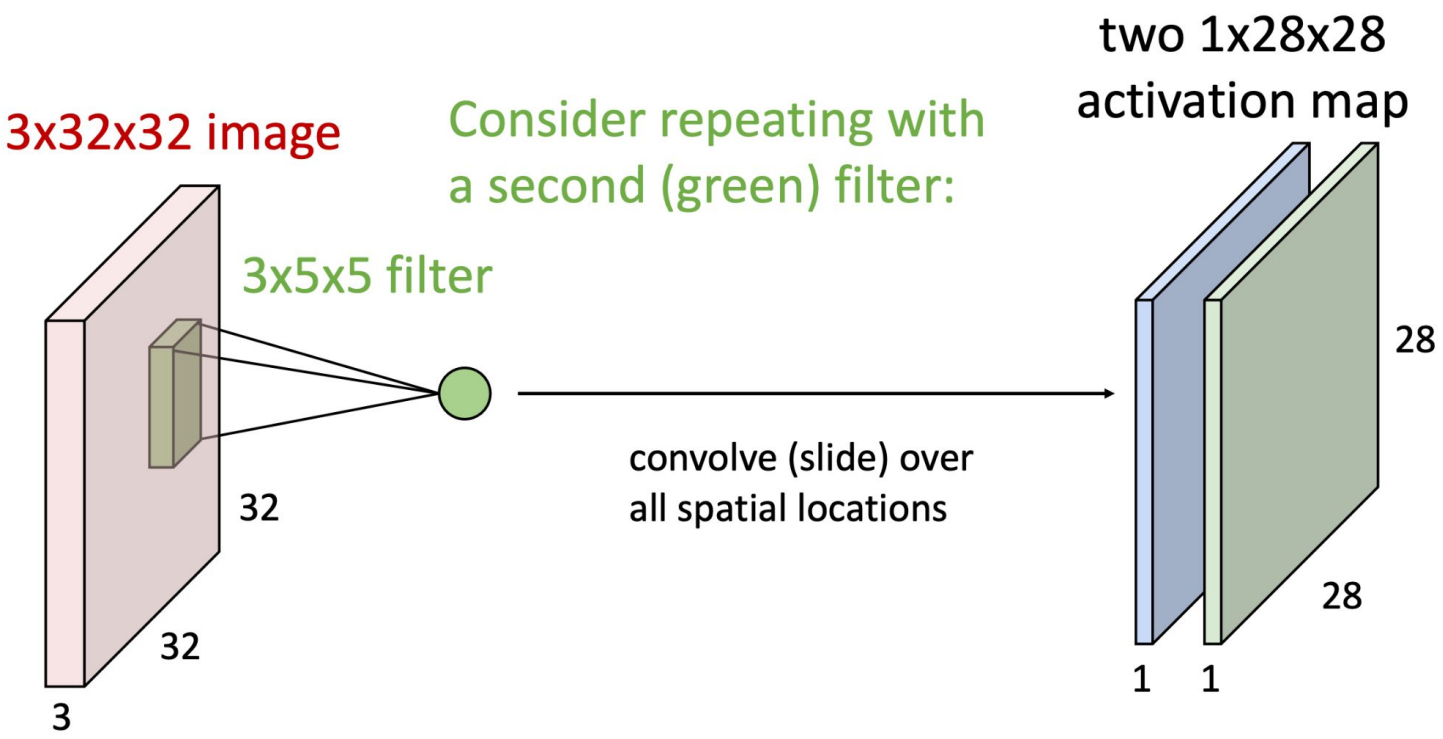
Convolution Layer



Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Convolutional Neural Network

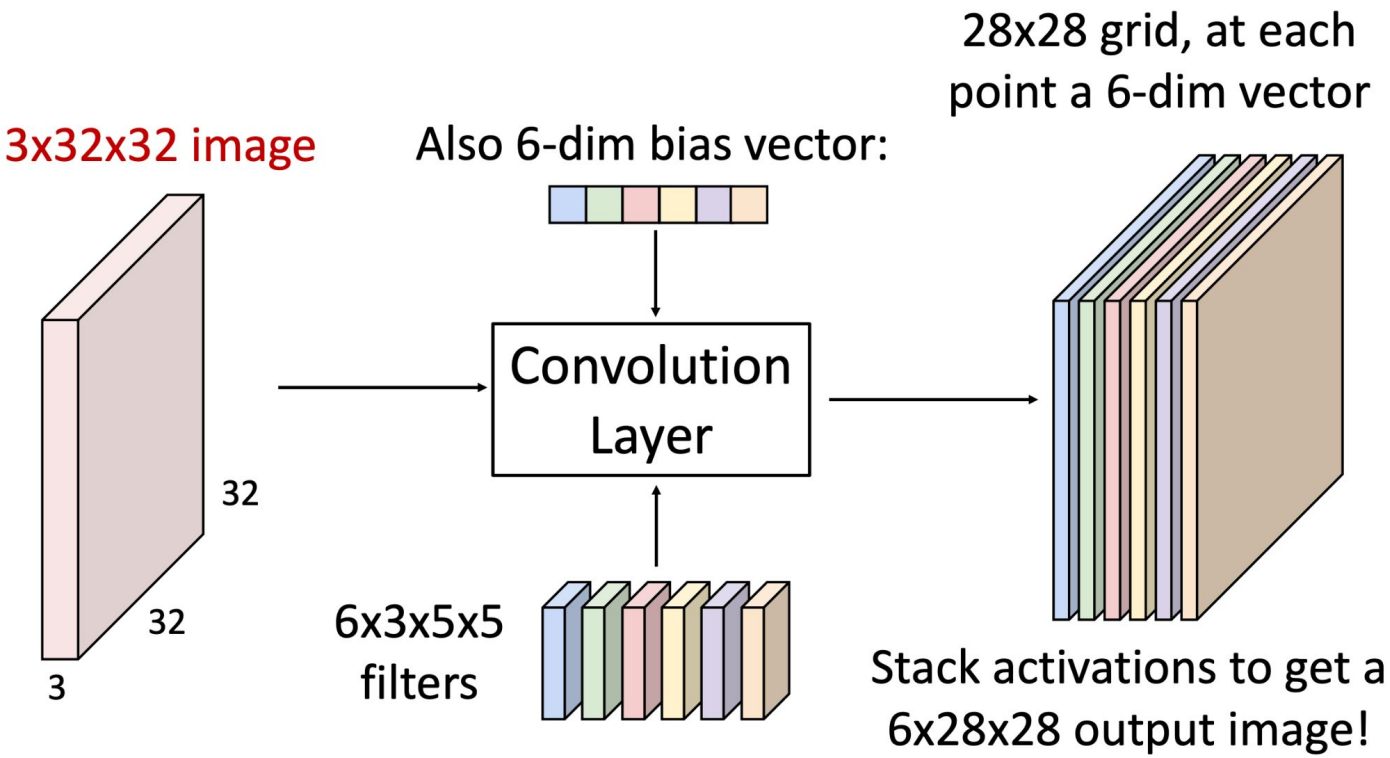
Convolution Layer



Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Convolutional Neural Network

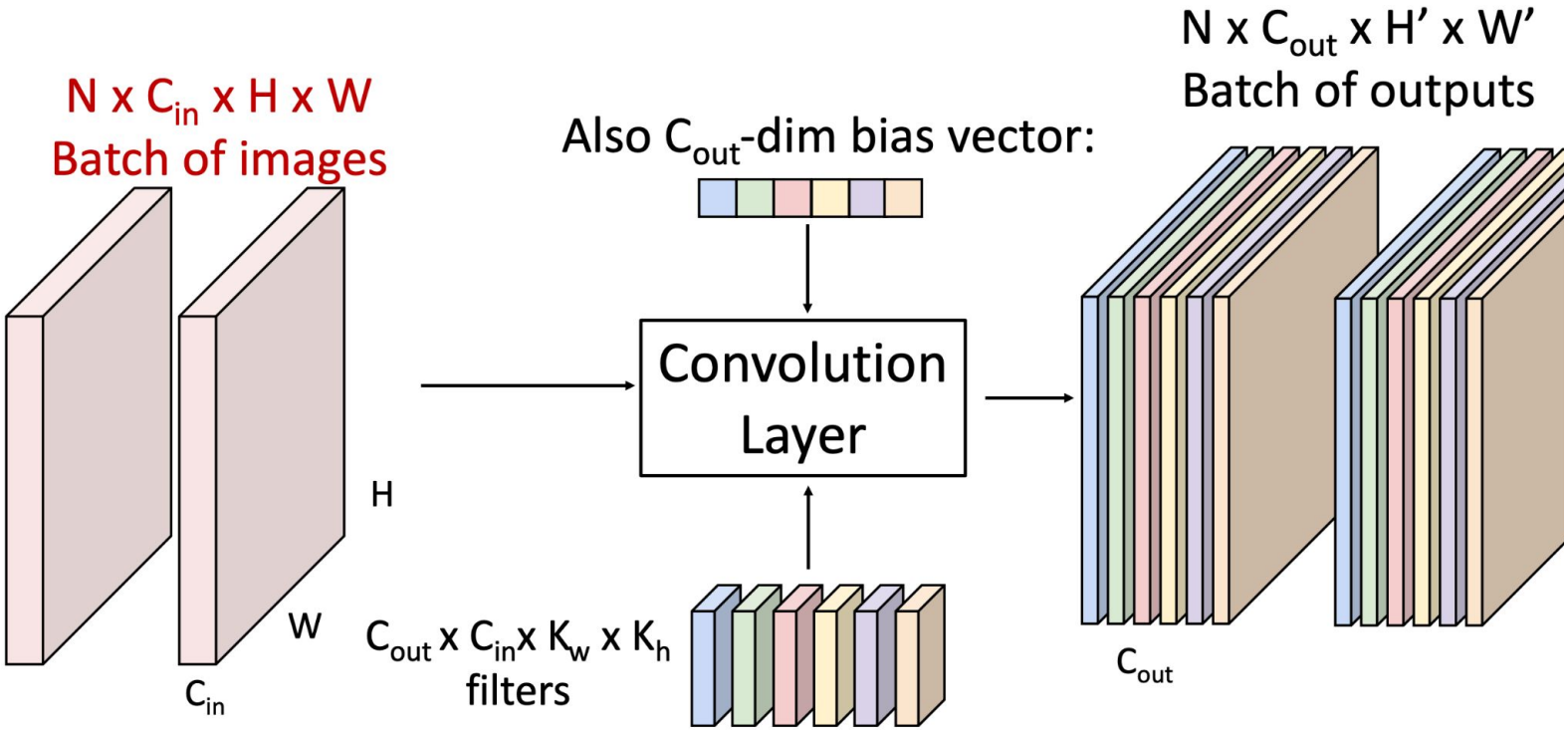
Convolution Layer



Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Convolutional Neural Network

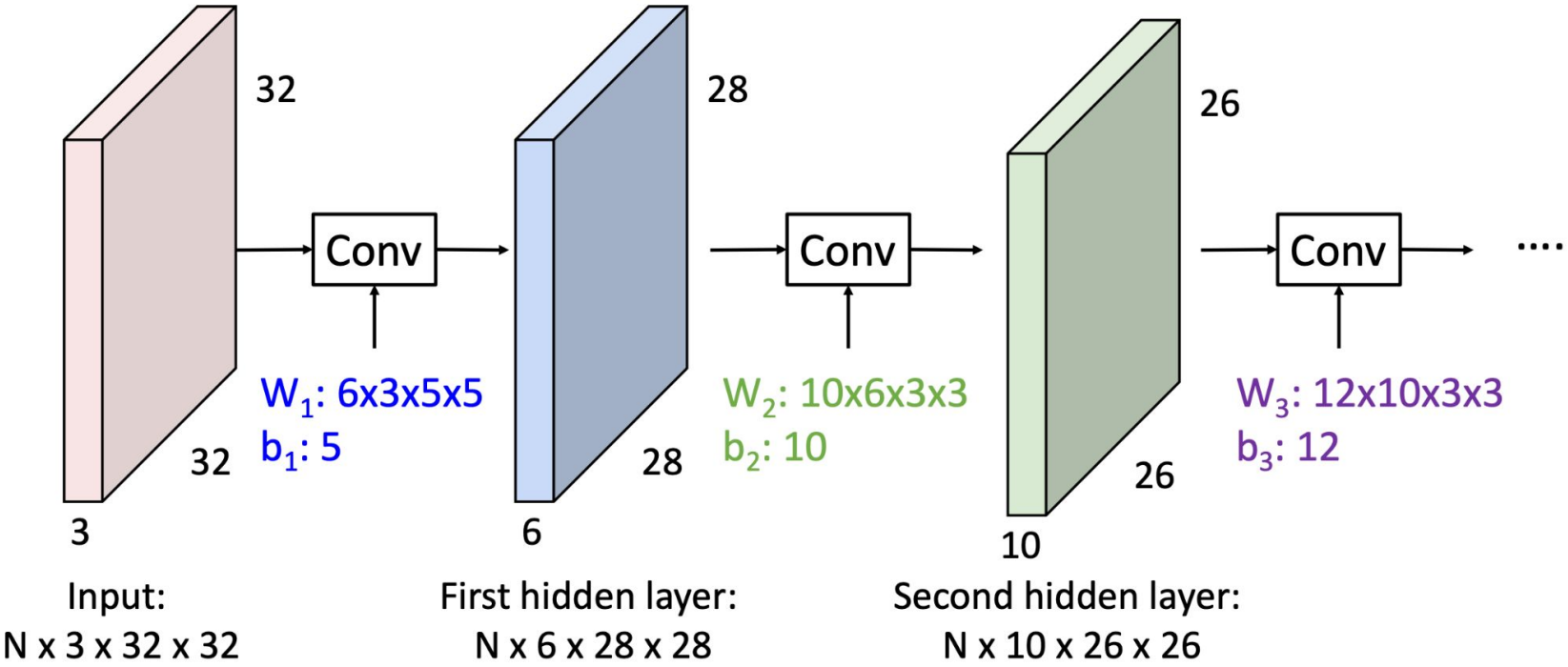
Convolution Layer



Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Convolutional Neural Network

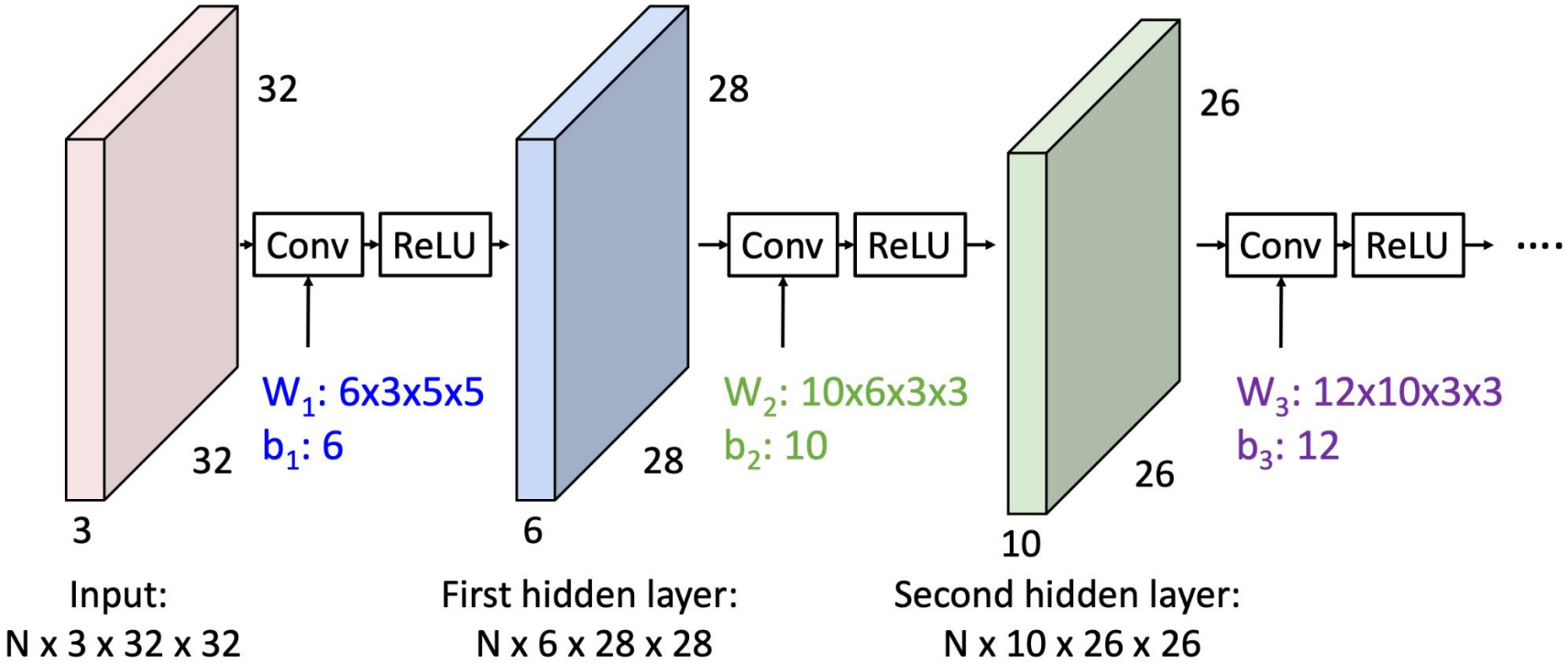
Stacking Convolutions



Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Convolutional Neural Network

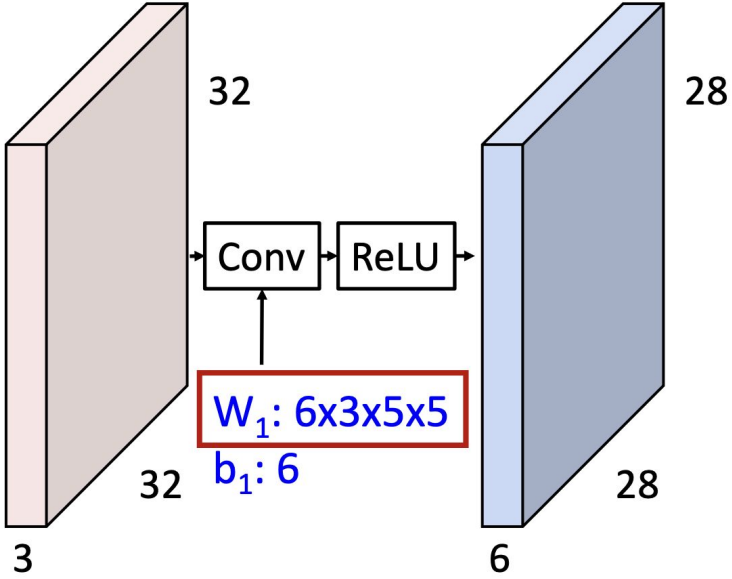
Stacking Convolutions



Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Convolutional Neural Network

What do convolutional filters learn ?



Input:
N x 3 x 32 x 32

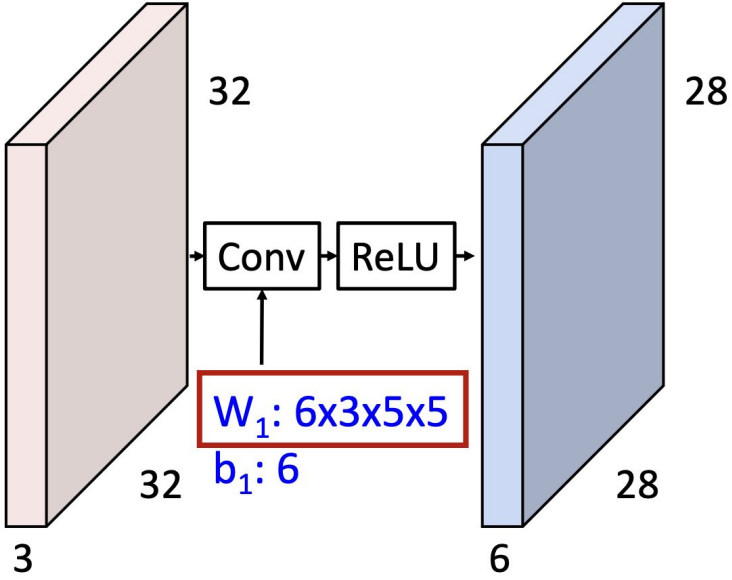
First hidden layer:
N x 6 x 28 x 28

Linear classifier: One template per class

Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Convolutional Neural Network

What do convolutional filters learn ?



Input:
N x 3 x 32 x 32

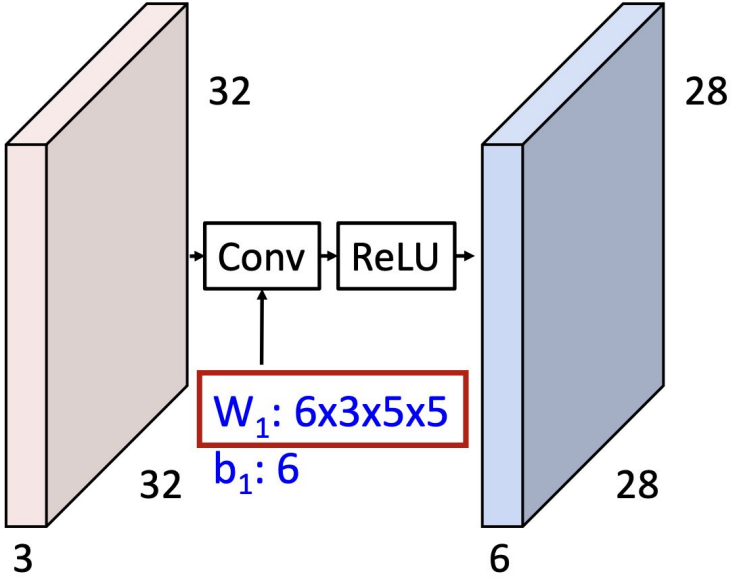
First hidden layer:
N x 6 x 28 x 28

MLP: Bank of whole-image templates

Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Convolutional Neural Network

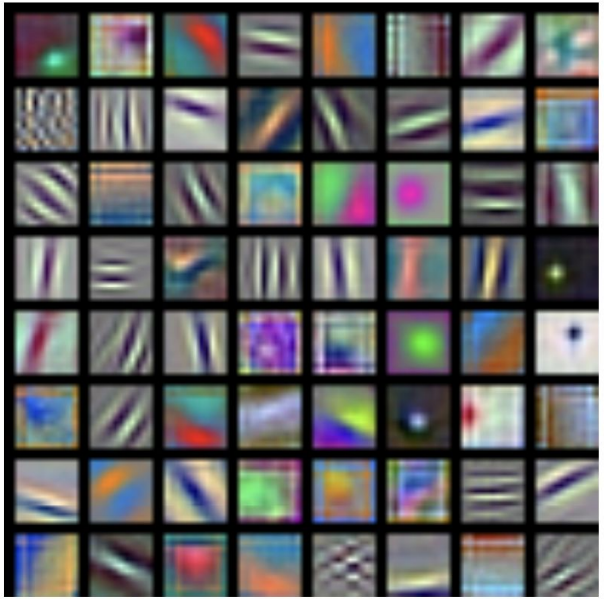
What do convolutional filters learn ?



Input:
N x 3 x 32 x 32

First hidden layer:
N x 6 x 28 x 28

First-layer conv filters: local image templates
(Often learns oriented edges, opposing colors)

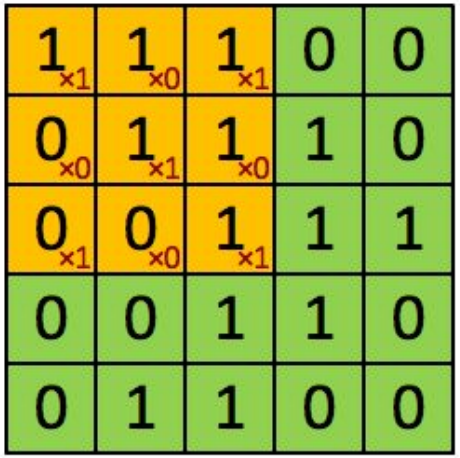


AlexNet: 64 filters, each 3x11x11

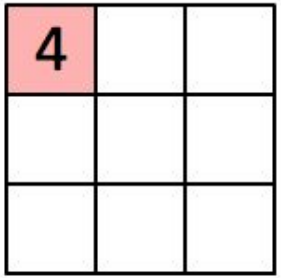
Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Convolutional Neural Network

Spatial Dimensions



Image



Convolved Feature

Input: 5x5

Filter: 3x3

Problem: Feature maps "shrink: with each layer!

In general:

Input: W

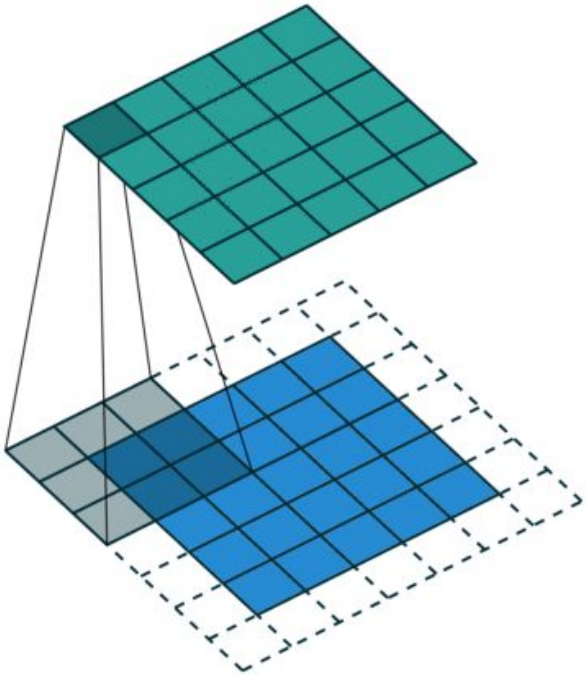
Filter: K

Output: W - K + 1

Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Convolutional Neural Network

Spatial Dimensions



Input: 5x5

Filter: 3x3

Padding: 1

Solution: **Padding**

Adding zeros

In general:

Input: W

Filter: K

Padding: P

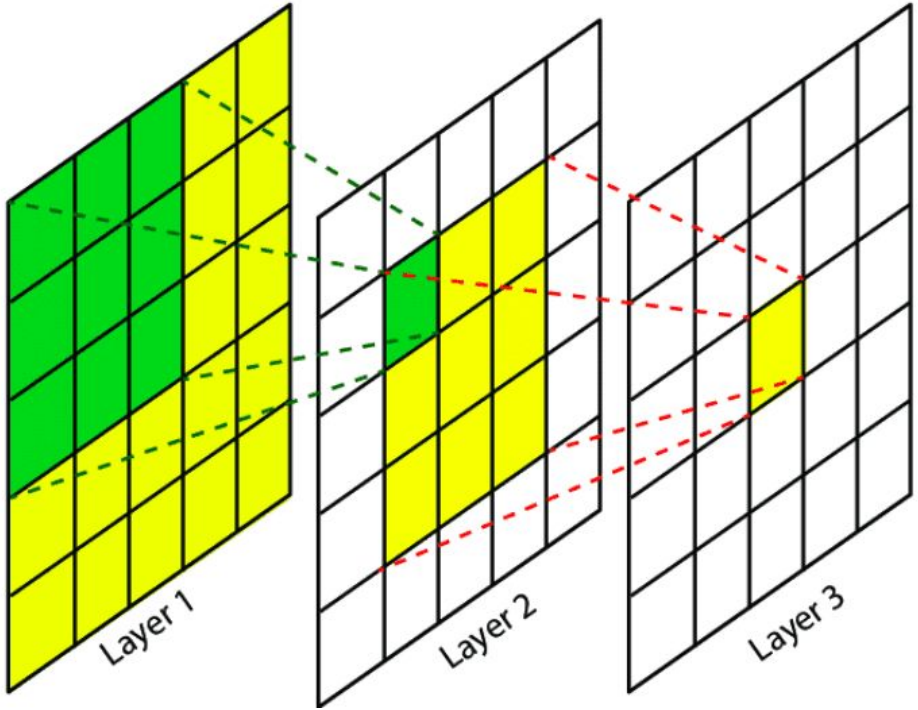
Output: W - K + 1 + 2P

Set P = (K - 1)/2 to make output have same size as input.

Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Convolutional Neural Network

Receptive Fields



For convolution with kernel size ***K***, each element in the output depends on a ***K x K*** receptive field in the input

Each successive conv. add ***K*** - 1 to the receptive field size.

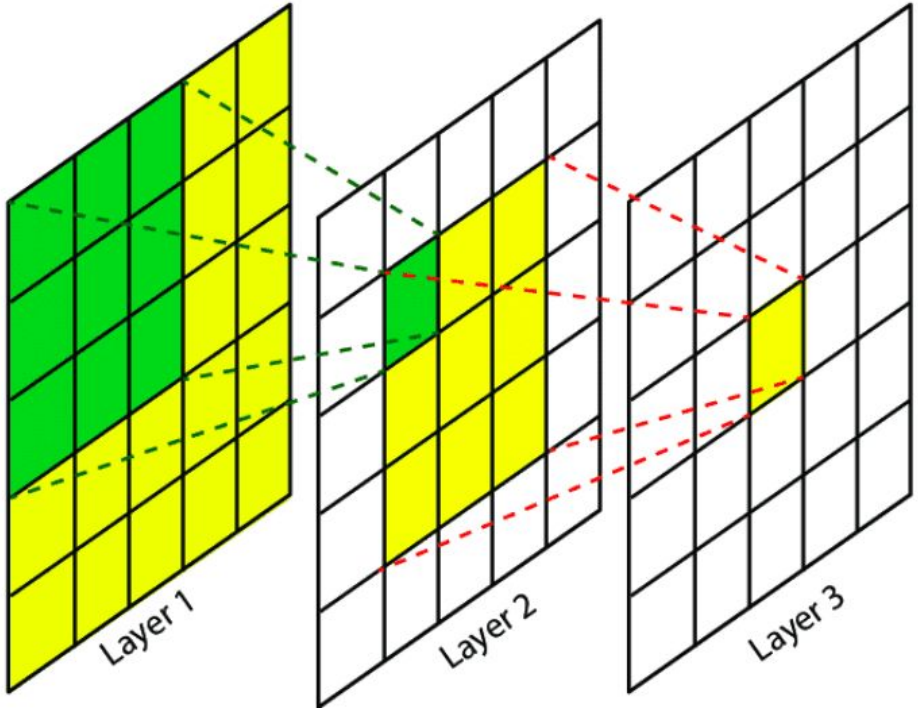
With ***L*** layers the receptive field size is 1 + ***L*** * (***K*** - 1)

ex : L=2, K = 3 then RF = 5

Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Convolutional Neural Network

Receptive Fields



For convolution with kernel size ***K***, each element in the output depends on a ***K x K*** receptive field in the input

Each successive conv. add ***K*** - 1 to the receptive field size.

With ***L*** layers the receptive field size is 1 + ***L*** * (***K*** - 1)

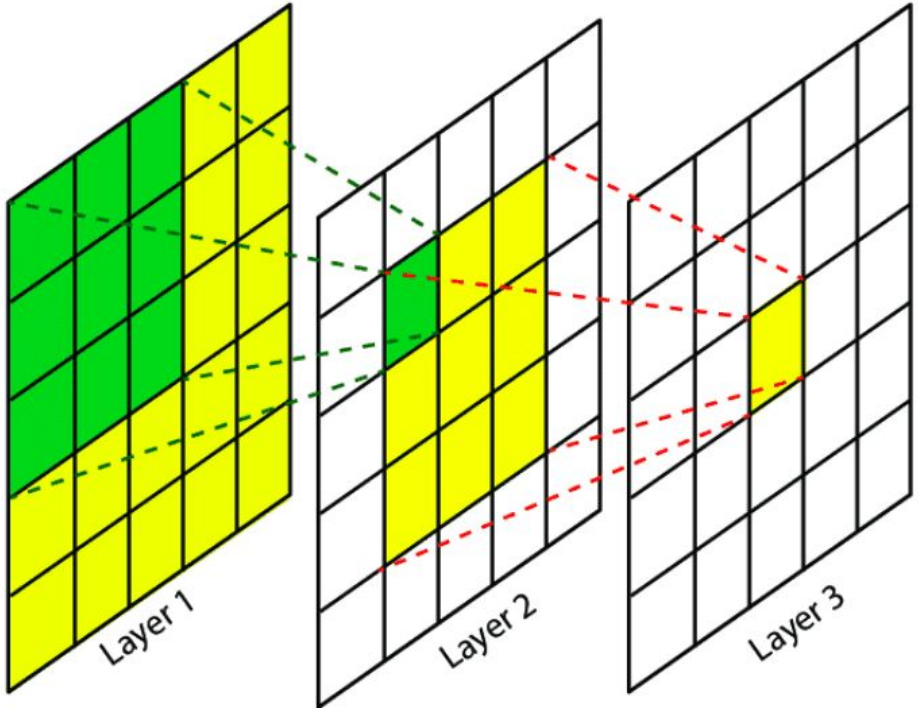
ex : L=2, K = 3 then RF = 5

Problem: For large images we need many layers for each output to "see" the whole image.

Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Convolutional Neural Network

Receptive Fields



For convolution with kernel size ***K***, each element in the output depends on a ***K x K*** receptive field in the input

Each successive conv. add ***K*** - 1 to the receptive field size.

With ***L*** layers the receptive field size is 1 + ***L*** * (***K*** - 1)

ex : L=2, K = 3 then RF = 5

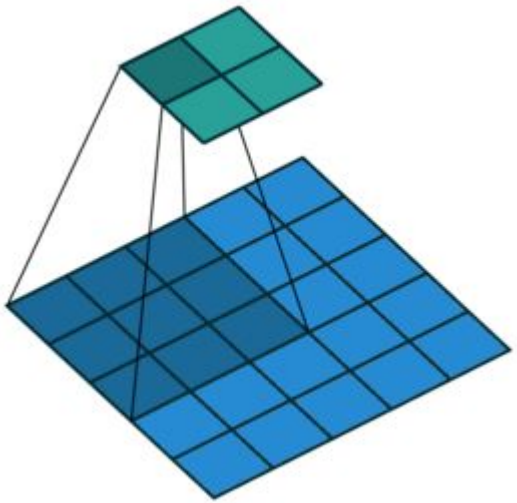
Problem: For large images we need many layers for each output to "see" the whole image.

Solution: Downsample inside the network

Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Convolutional Neural Network

Strided Convolution



Input: 5x5

Filter: 3x3

Padding: 0

Stride: 2

Output: 2x2

In general:

Input: W

Filter: K

Padding: P

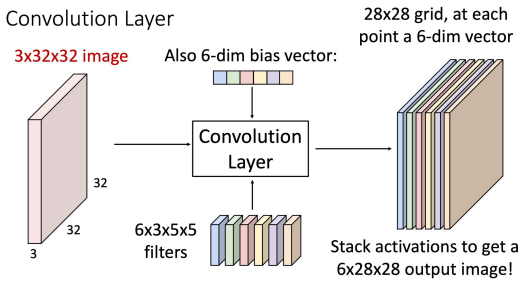
Stride: S

Output: (W - K + 2P) / S + 1

Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Convolutional Neural Network

Example



Output volume size: ?

Input: 3 x 32 x 32

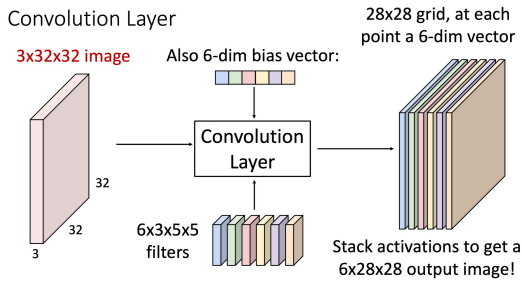
10 of Filters with size 5x5

Padding: 2, Stride: 1

Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Convolutional Neural Network

Example



Output volume size: ?

(32 - 5 + (2*2))/1 + 1 = 32

10 x 32 x 32

Input: 3 x 32 x 32

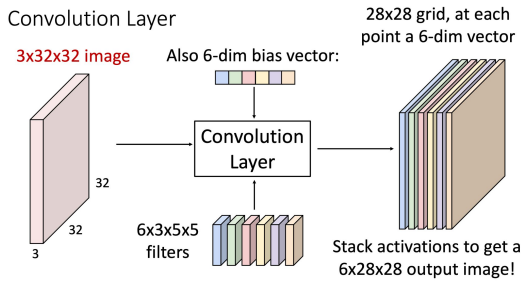
10 of Filters with size 5x5

Padding: 2, Stride: 1

Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Convolutional Neural Network

Example



Output volume size: ?

(32 - 5 + (2*2))/1 + 1 = 32

10 x 32 x 32

Number of learnable parameters: ?

Input: 3 x 32 x 32

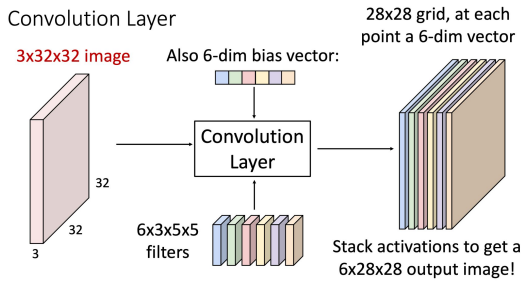
10 of Filters with size 5x5

Padding: 2, Stride: 1

Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Convolutional Neural Network

Example



Input: 3 x 32 x 32

10 of Filters with size 5x5

Padding: 2, Stride: 1

Output volume size: ?

(32 - 5 + (2*2))/1 + 1 = 32

10 x 32 x 32

Number of learnable parameters: ?

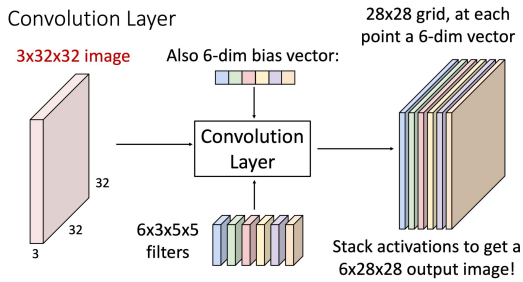
(3x5x5) + 1 = 76

10 filters => 10 * 76 = 760

Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Convolutional Neural Network

Example



Input: 3 x 32 x 32

10 of Filters with size 5x5

Padding: 2, Stride: 1

Output volume size: ?

(32 - 5 + (2*2))/1 + 1 = 32

10 x 32 x 32

Number of learnable parameters: ?

(3x5x5) + 1 = 76

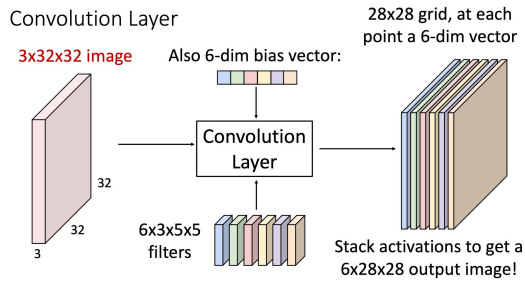
10 filters => 10 * 76 = 760

Number of Multiply-add operations: ?

Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Convolutional Neural Network

Example



Input: 3 x 32 x 32

10 of Filters with size 5x5

Padding: 2, Stride: 1

Output volume size: ?

(32 - 5 + (2*2))/1 + 1 = 32

10 x 32 x 32

Number of learnable parameters: ?

(3x5x5) + 1 = 76

10 filters => 10 * 76 = 760

Number of Multiply-add operations: ?

10x32x32 = 10,240 outputs

Each output is the inner product of two 3x5x5 tensors (75 elems)

75 x 10,240 = 768,000

Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

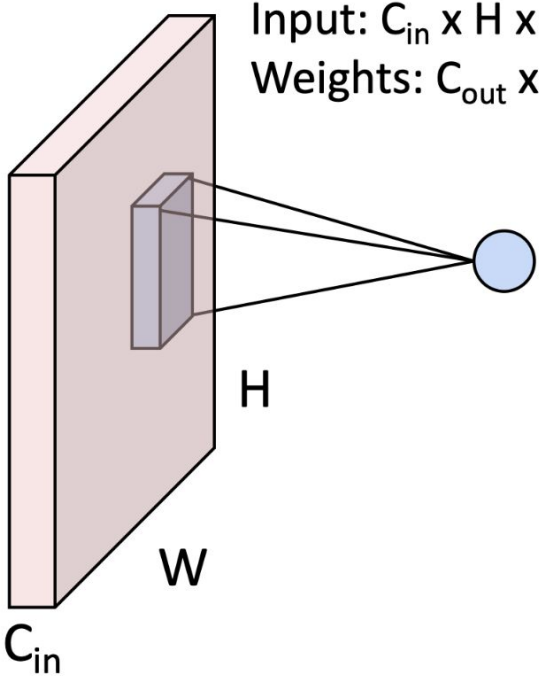
Convolutional Neural Network

Other types of convolution

2D Convolution

Input: C_{in} x H x W

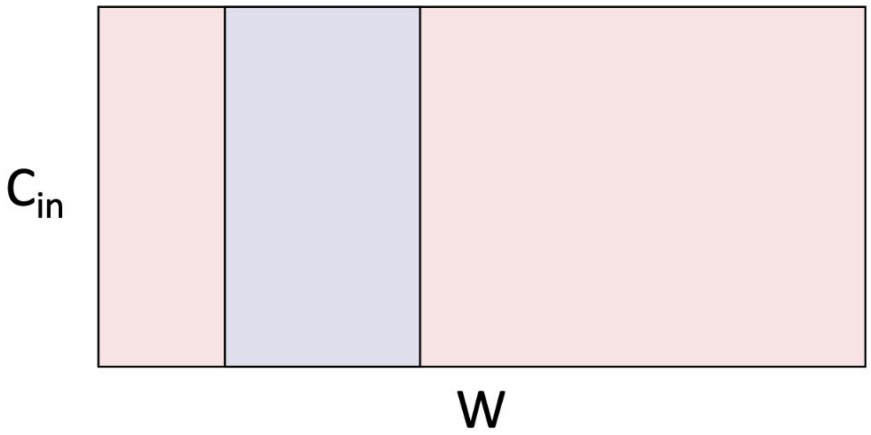
Weights: C_{out} x C_{in} x K x K



1D Convolution

Input: C_{in} x W

Weights: C_{out} x C_{in} x K



Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

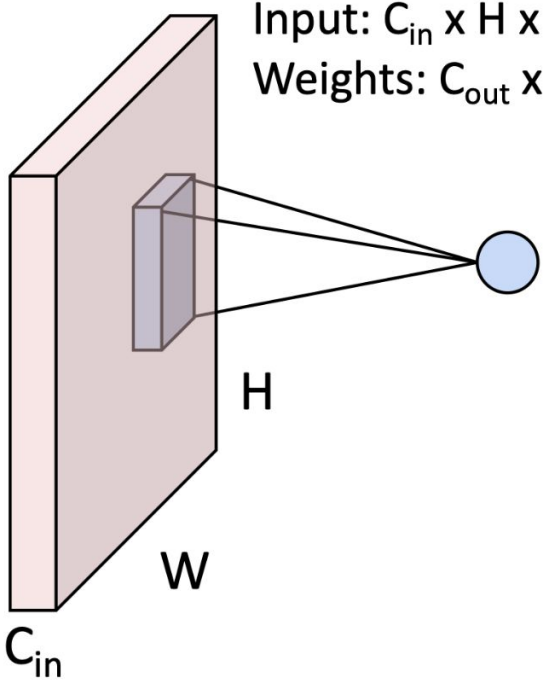
Convolutional Neural Network

Other types of convolution

2D Convolution

Input: C_{in} x H x W

Weights: C_{out} x C_{in} x K x K

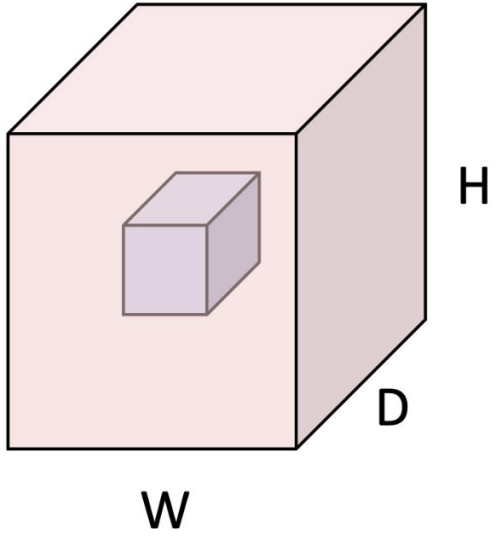


3D Convolution

Input: C_{in} x H x W x D

Weights: C_{out} x C_{in} x K x K x K

C_{in}-dim vector at each point in the volume

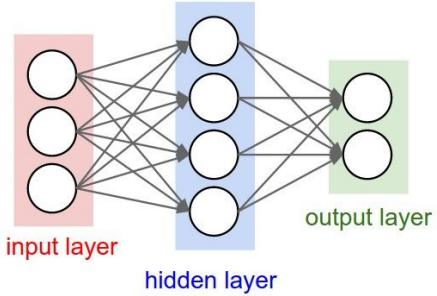


Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

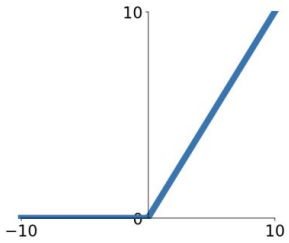
Convolutional Neural Network

Components of a Convolutional Network

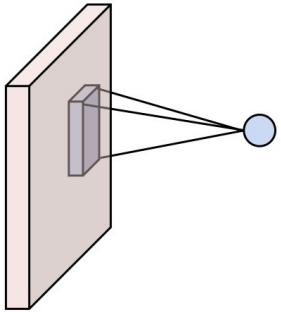
Fully-Connected Layer



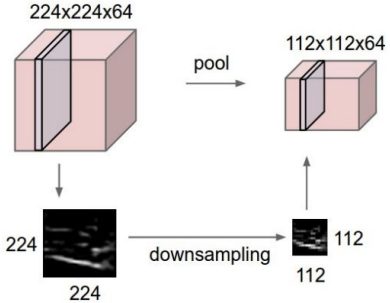
Activation Function



Convolution Layers



Pooling Layers



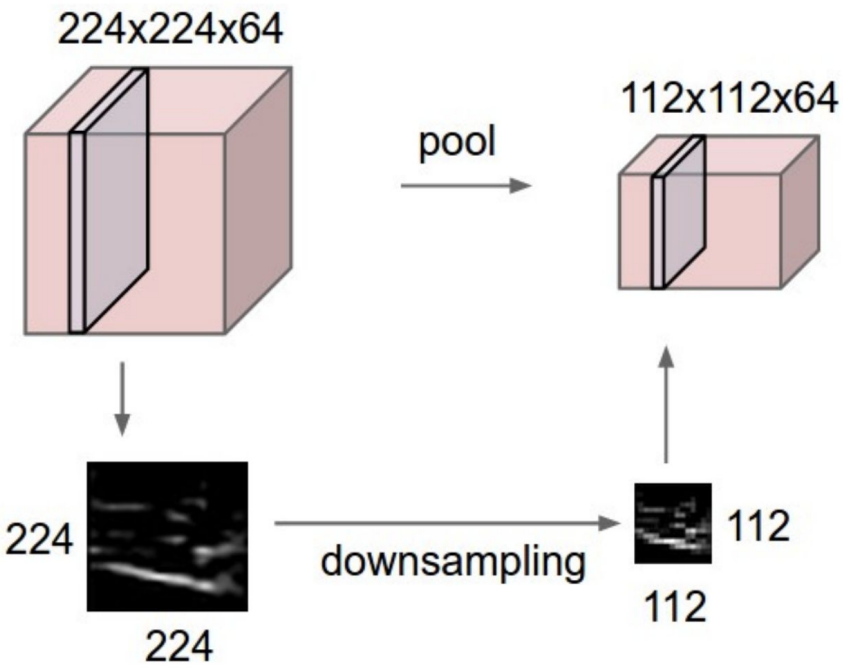
Normalization

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Convolutional Neural Network

Pooling Layers: Another way to downsample



Hyperparameters:

Kernel size

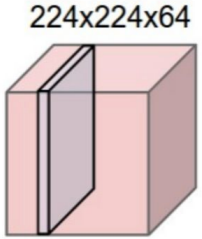
Stride

Pooling function

Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Convolutional Neural Network

Maxpooling



Single depth slice

x

1	1	2	4
5	**6**	7	**8**
3	2	1	0
1	2	3	**4**

y

Max pooling with 2x2 kernel size and stride 2

6	8
3	4

Introduces **invariance** to small spatial shifts

No learnable parameters!

Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Convolutional Neural Network

Pooling Layers: Another way to downsample

Input: C x H x W

Hyperparameters:

- **Kernel size:** K
- **Stride:** S
- **Pooling function** (max, avg)

Output: C x H' x W' where

- H' = (H – K) / S + 1
- W' = (W – K) / S + 1

Learnable parameters: **None!**

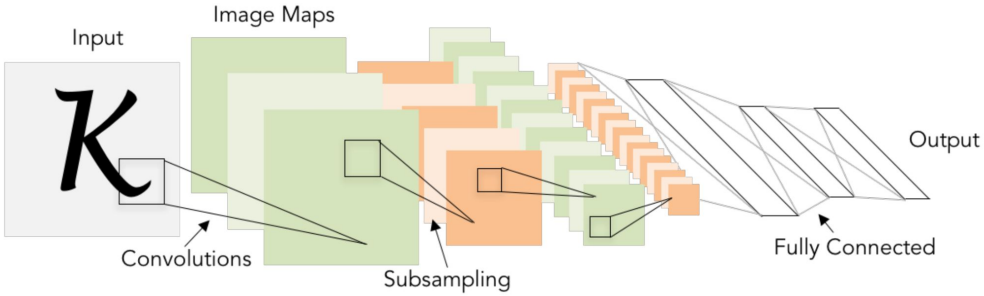
Common settings:
Max, K =2, S = 2
Max, K = 3, S = 2

Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

Convolutional Neural Network

Architecture Example

Layer	Output Size	Weight Size
Input	1 x 28 x 28	
Conv (C_{out}=20, K=5, P=2, S=1)	20 x 28 x 28	20 x 1 x 5 x 5
ReLU	20 x 28 x 28	
MaxPool(K=2, S=2)	20 x 14 x 14	
Conv (C_{out}=50, K=5, P=2, S=1)	50 x 14 x 14	50 x 20 x 5 x 5
ReLU	50 x 14 x 14	
MaxPool(K=2, S=2)	50 x 7 x 7	
Flatten	2450	
Linear (2450 -> 500)	500	2450 x 500
ReLU	500	
Linear (500 -> 10)	10	500 x 10

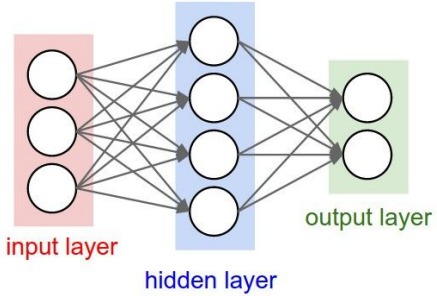


Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

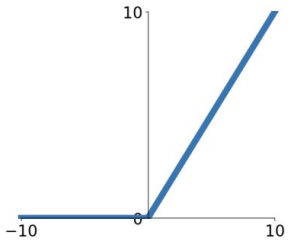
Convolutional Neural Network

Components of a Convolutional Network

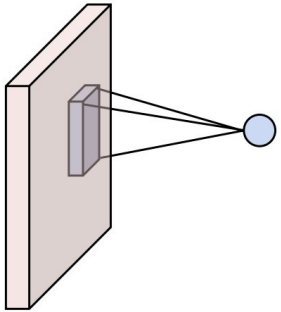
Fully-Connected Layer



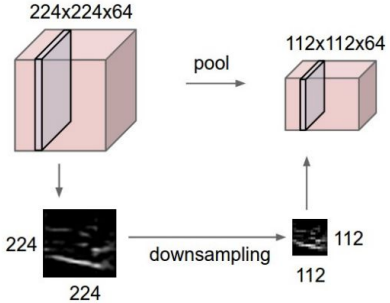
Activation Function



Convolution Layers



Pooling Layers



Normalization

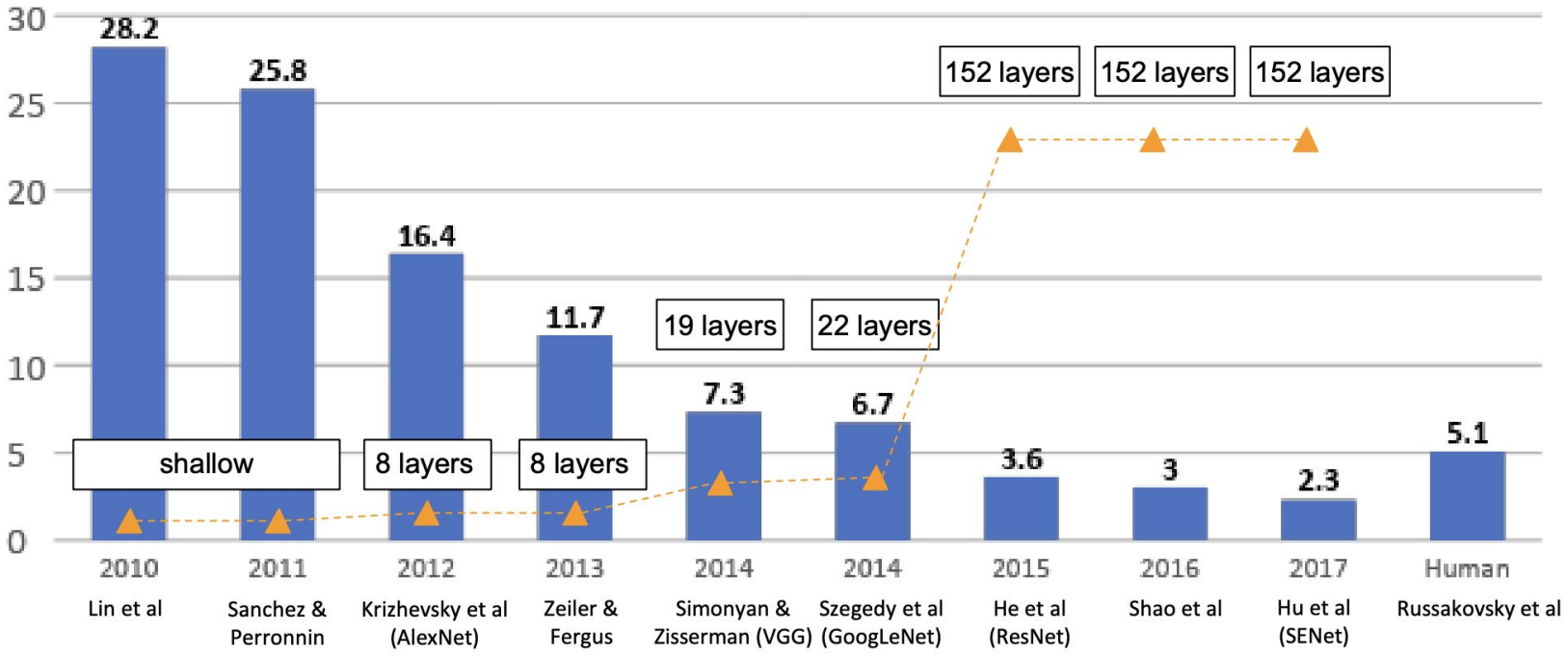
$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Slide adopted from: Justin Johnson, EECS 498-007 / 598-005: Deep Learning for Computer Vision, University of Michigan, https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

CNN
Architectures

Convolutional Neural Network

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



Slide adopted from: Fei Fei Li, CS231n: Deep Learning for Computer Vision, Stanford University, https://cs231n.stanford.edu/

Convolutional Neural Network

AlexNet

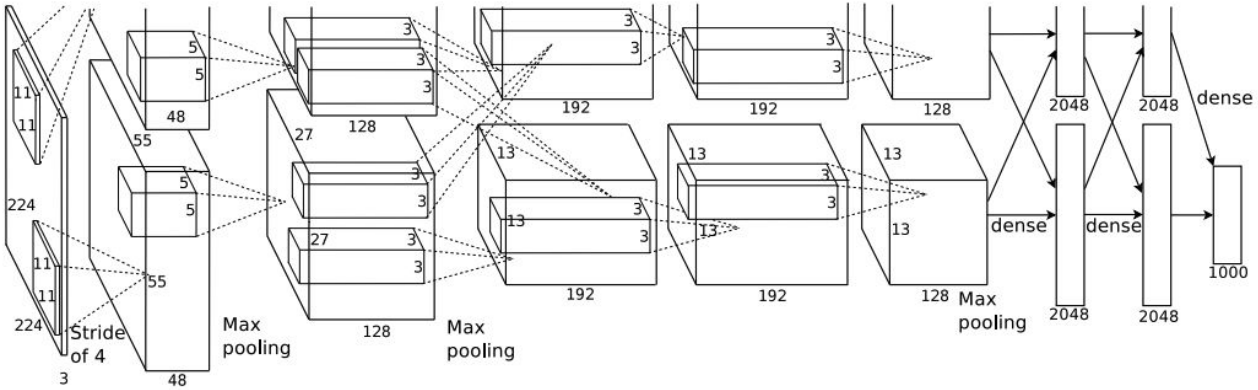
MaxPool_1
Input: 55x55x96
Filter: 3x3
Stride: 2
Output: 55x55x96

MaxPool_2
Input: 27x27x256
Filter: 3x3
Stride: 2
Output: 27x27x256

MaxPool_3
Input: 13x13x256
Filter: 3x3
Stride: 2
Output: 6x6x256

FC_1
Input: 4096
Output: 4096

FC_2
Input: 4096
Output: 4096



FC_3
Input: 4096
Output: 1000

Conv_1
Input: 224x224x3
Filter: 96 11x11x3
Padding: 0
Stride: 4
Output: 55x55x96

Conv_2
Input: 55x55x96
Filter: 256 5x5x96
Padding: 2
Stride: 1
Output: 27x27x256

Conv_3
Input: 27x27x256
Filter: 384 3x3x256
Padding: 1
Stride: 1
Output: 13x13x384

Conv_4
Input: 13x13x384
Filter: 384 3x3x384
Padding: 1
Stride: 1
Output: 13x13x384

Conv_5
Input: 13x13x384
Filter: 256 3x3x384
Padding: 1
Stride: 1
Output: 13x13x256

Slide adopted from: Fei Fei Li, CS231n: Deep Learning for Computer Vision, Stanford University, https://cs231n.stanford.edu/

Convolutional Neural Network

AlexNet

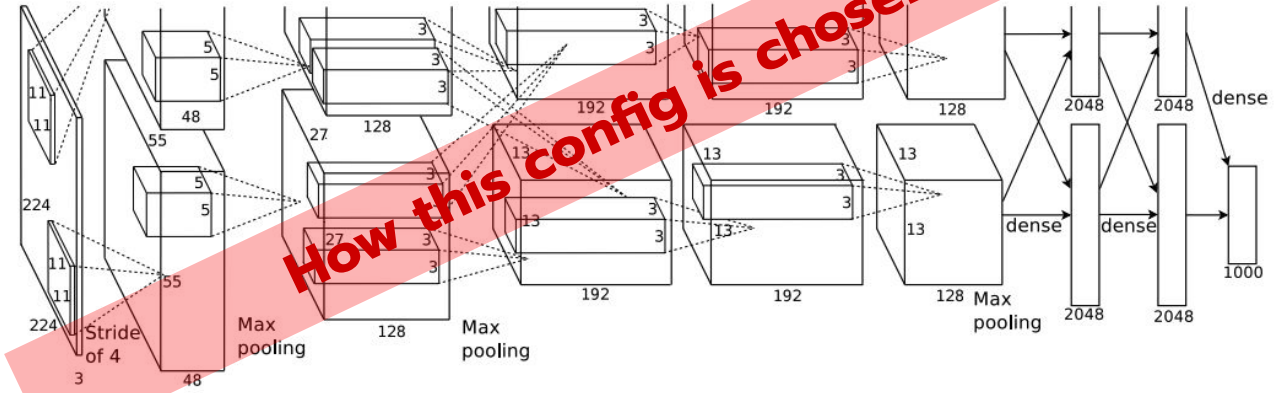
MaxPool_1
Input: 55x55x96
Filter: 3x3
Stride: 2
Output: 55x55x96

MaxPool_2
Input: 27x27x256
Filter: 3x3
Stride: 2
Output: 27x27x256

MaxPool_3
Input: 13x13x256
Filter: 3x3
Stride: 2
Output: 6x6x256

FC_1
Input: 4096
Output: 4096

FC_2
Input: 4096
Output: 4096



FC_3
Input: 4096
Output: 1000

Conv_1
Input: 224x224x3
Filter: 96 11x11x3
Padding: 0
Stride: 4
Output: 55x55x96

Conv_2
Input: 55x55x96
Filter: 256 5x5x96
Padding: 2
Stride: 1
Output: 27x27x256

Conv_3
Input: 27x27x256
Filter: 384 3x3x256
Padding: 1
Stride: 1
Output: 13x13x384

Conv_4
Input: 13x13x384
Filter: 384 3x3x384
Padding: 1
Stride: 1
Output: 13x13x384

Conv_5
Input: 13x13x384
Filter: 256 3x3x384
Padding: 1
Stride: 1
Output: 13x13x256

Slide adopted from: Fei Fei Li, CS231n: Deep Learning for Computer Vision, Stanford University, https://cs231n.stanford.edu/

Convolutional Neural Network

AlexNet

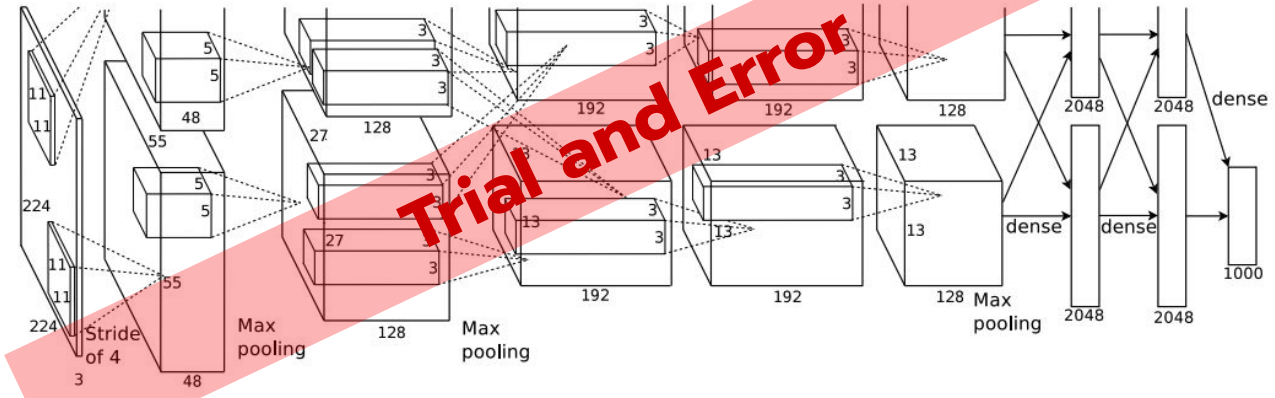
MaxPool_1
Input: 55x55x96
Filter: 3x3
Stride: 2
Output: 55x55x96

MaxPool_2
Input: 27x27x256
Filter: 3x3
Stride: 2
Output: 27x27x256

MaxPool_3
Input: 13x13x256
Filter: 3x3
Stride: 2
Output: 6x6x256

FC_1
Input: 4096
Output: 4096

FC_2
Input: 4096
Output: 4096



FC_3
Input: 4096
Output: 1000

Conv_1
Input: 224x224x3
Filter: 96 11x11x3
Padding: 0
Stride: 4
Output: 55x55x96

Conv_2
Input: 55x55x96
Filter: 256 5x5x96
Padding: 2
Stride: 1
Output: 27x27x256

Conv_3
Input: 27x27x256
Filter: 384 3x3x256
Padding: 1
Stride: 1
Output: 13x13x384

Conv_4
Input: 13x13x384
Filter: 384 3x3x384
Padding: 1
Stride: 1
Output: 13x13x384

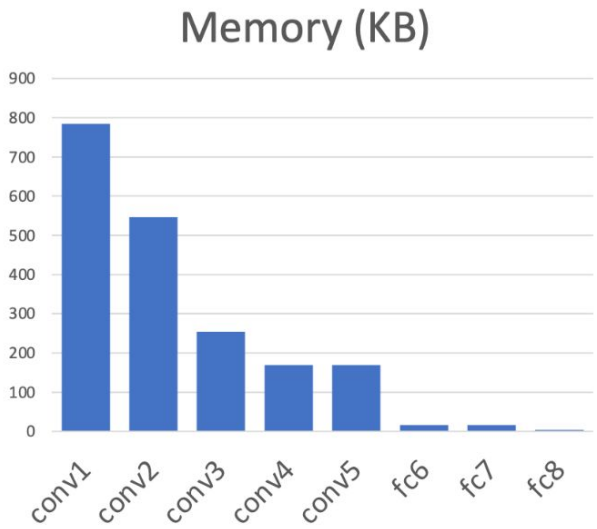
Conv_5
Input: 13x13x384
Filter: 256 3x3x384
Padding: 1
Stride: 1
Output: 13x13x256

Slide adopted from: Fei Fei Li, CS231n: Deep Learning for Computer Vision, Stanford University, https://cs231n.stanford.edu/

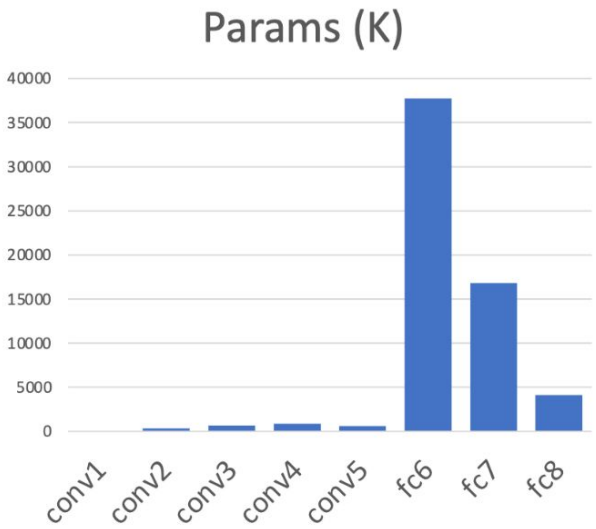
Convolutional Neural Network

AlexNet

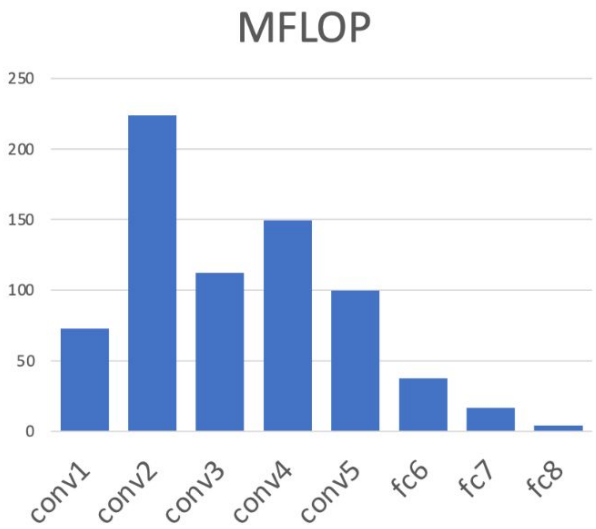
Most of the **memory usage** is in the early convolution layers



Nearly all **parameters** are in the fully-connected layers



Most **floating-point ops** occur in the convolution layers



Slide adopted from: Fei Fei Li, CS231n: Deep Learning for Computer Vision, Stanford University, https://cs231n.stanford.edu/

Convolutional Neural Network

VGGNet

VGG Design rules:

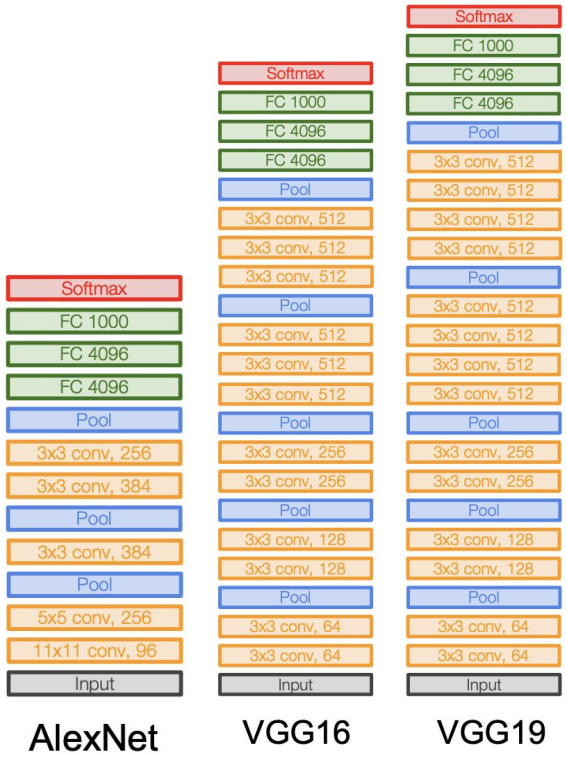
All conv are 3x3 stride 1 pad 1
All max pool are 2x2 stride 2
After pool, double #channels

Small filters, Deeper networks

8 layers (AlexNet) -> 16 - 19 layers (VGG16Net)

Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

Q: Why use smaller filters? (3x3 conv)

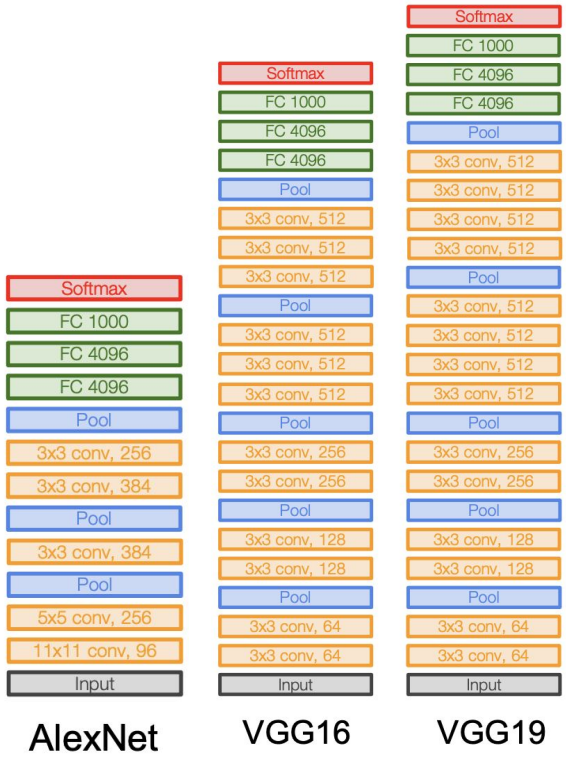


Slide adopted from: Fei Fei Li, CS231n: Deep Learning for Computer Vision, Stanford University, https://cs231n.stanford.edu/

Convolutional Neural Network

VGGNet

Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?

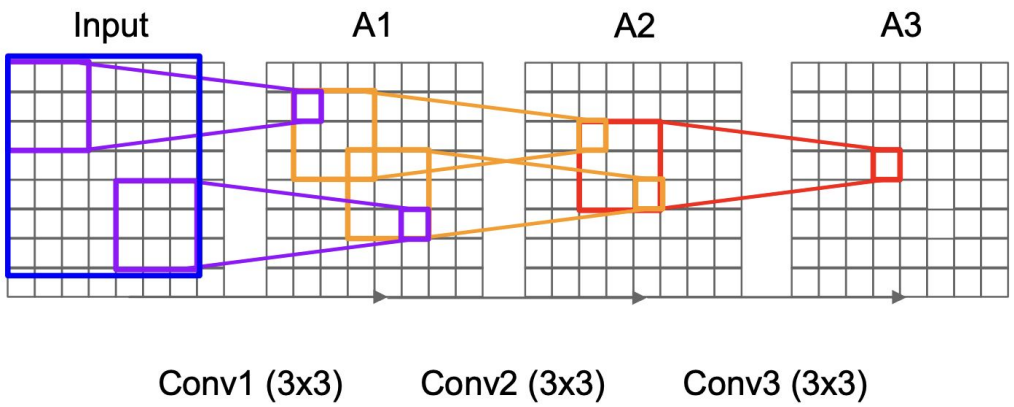


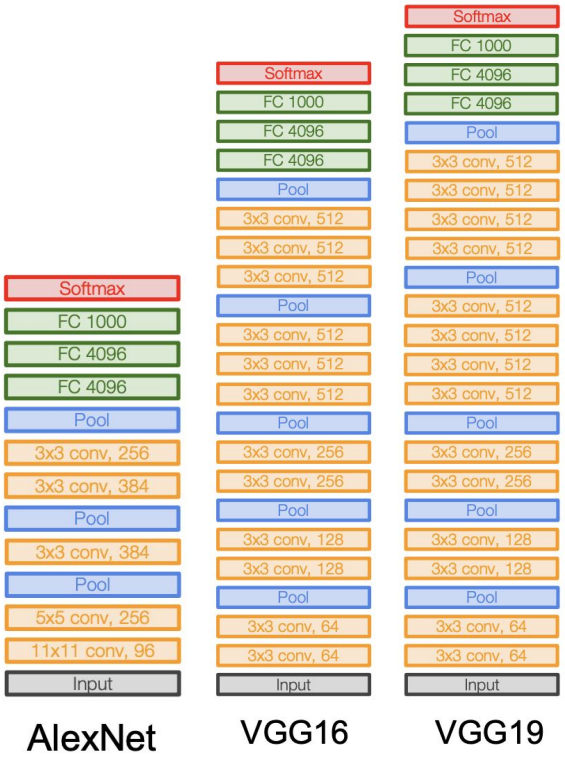
Slide adopted from: Fei Fei Li, CS231n: Deep Learning for Computer Vision, Stanford University, https://cs231n.stanford.edu/

Convolutional Neural Network

VGGNet

Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?





Stack of three 3x3 conv (stride 1) layers has same effective receptive field as one 7x7 conv layer

Slide adopted from: Fei Fei Li, CS231n: Deep Learning for Computer Vision, Stanford University, https://cs231n.stanford.edu/

Convolutional Neural Network

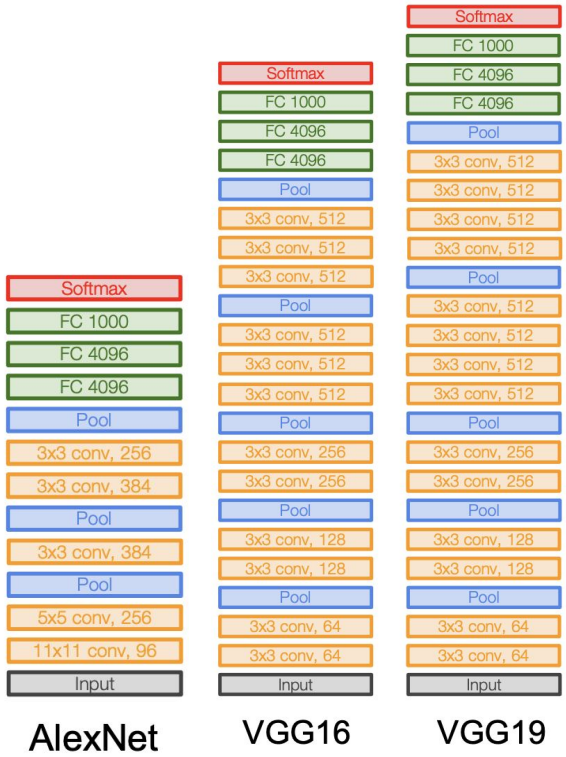
VGGNet

Q: Why use smaller filters? (3x3 conv)

But deeper, **more non-linearities**

And **fewer parameters**:

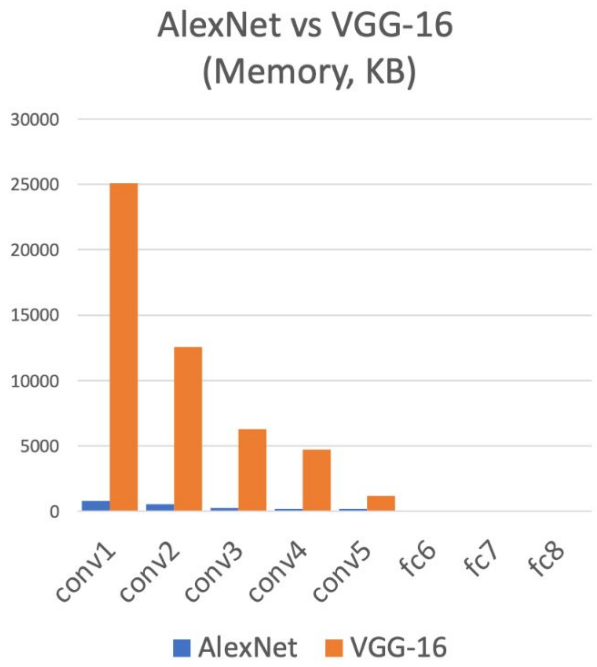
$3 * (3^2C^2)$ vs. 7^2C^2
for C channels per layer

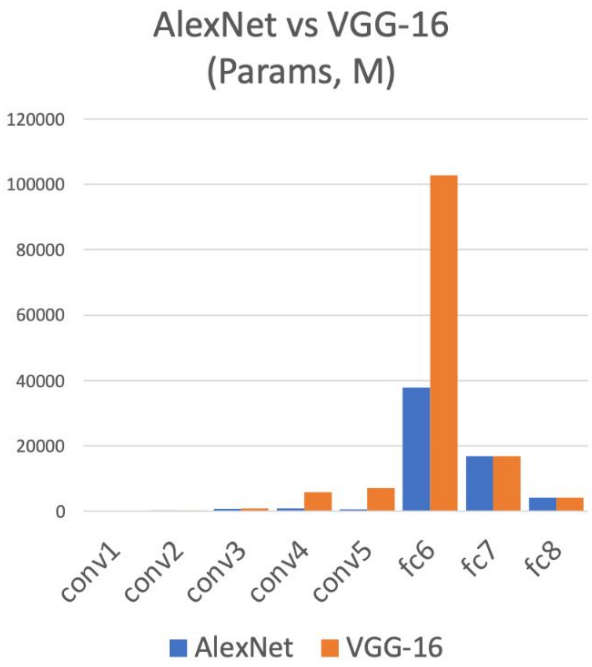


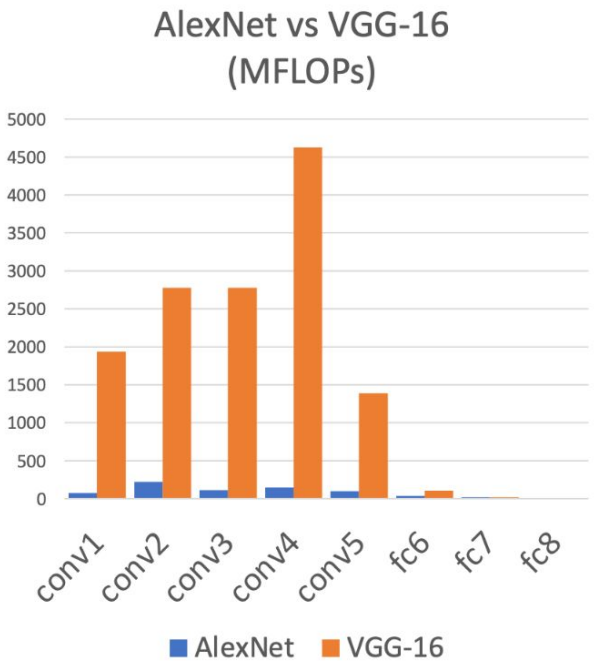
Slide adopted from: Fei Fei Li, CS231n: Deep Learning for Computer Vision, Stanford University, https://cs231n.stanford.edu/

Convolutional Neural Network

AlexNet vs. VGG-16 : Much Bigger Network







AlexNet total: 1.9 MB
VGG-16 total: 48.6 MB (25x)

AlexNet total: 61M
VGG-16 total: 138M (2.3x)

AlexNet total: 0.7 GFLOP
VGG-16 total: 13.6 GFLOP (19.4x)

Slide adopted from: Fei Fei Li, CS231n: Deep Learning for Computer Vision, Stanford University, https://cs231n.stanford.edu/

Convolutional Neural Network

ResNet

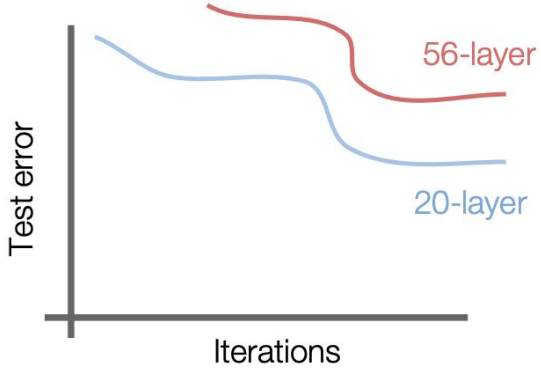
What happens when we continue **stacking deeper** layers on a **"plain"** convolutional neural network?

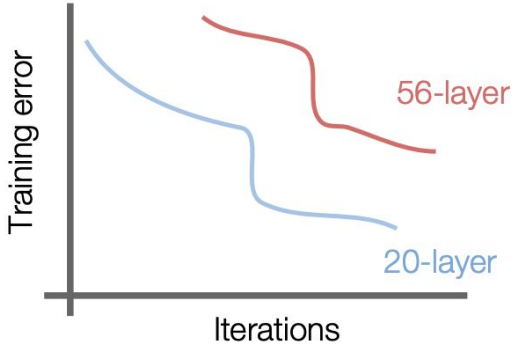
Slide adopted from: Fei Fei Li, CS231n: Deep Learning for Computer Vision, Stanford University, https://cs231n.stanford.edu/

Convolutional Neural Network

ResNet

What happens when we continue **stacking deeper** layers on a **"plain"** convolutional neural network?





56-layer model performs **worse** on both test and training error
-> The **deeper** model performs **worse**, but it's not caused by overfitting!

Slide adopted from: Fei Fei Li, CS231n: Deep Learning for Computer Vision, Stanford University, https://cs231n.stanford.edu/

Convolutional Neural Network

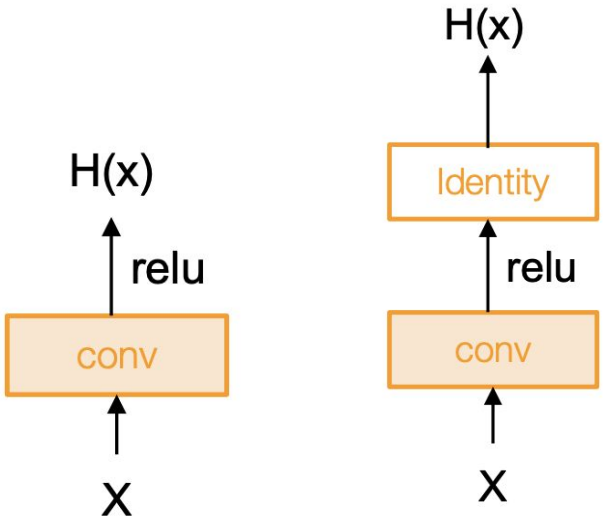
ResNet

Fact: Deep models have more representation power (more parameters) than shallower models.

Hypothesis: the problem is an optimization problem, deeper models are harder to optimize

Q: What should the deeper model learn to be at least as good as the shallower model?

Solution: Change the network so learning identity functions with extra layers is easy

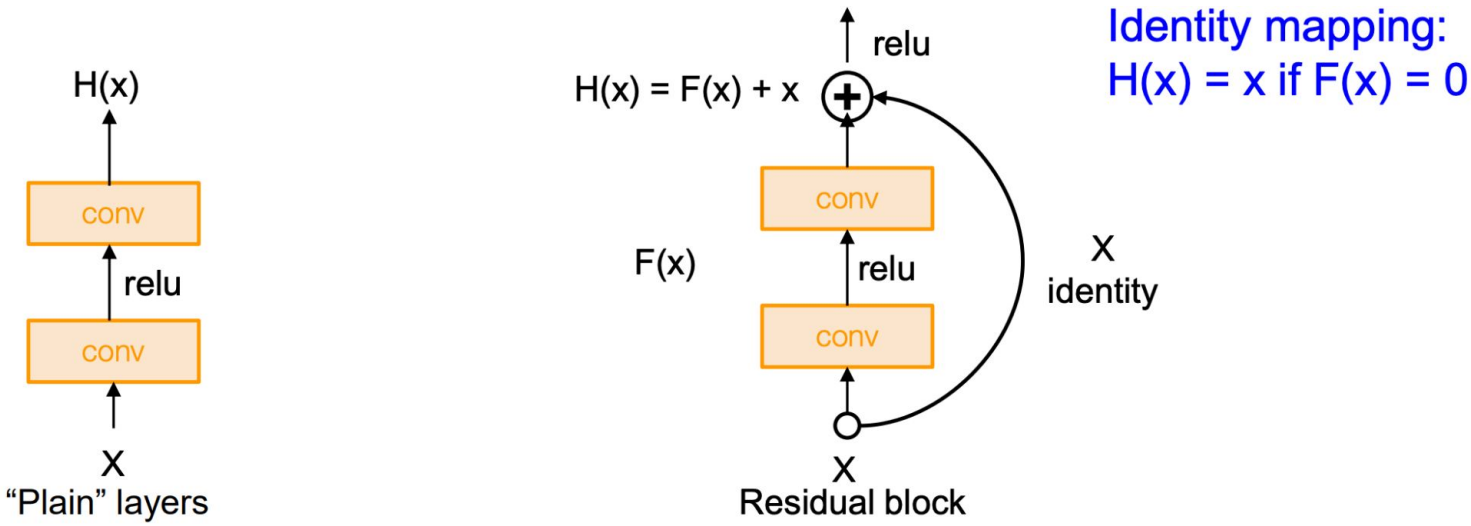


Slide adopted from: Fei Fei Li, CS231n: Deep Learning for Computer Vision, Stanford University, https://cs231n.stanford.edu/

Convolutional Neural Network

ResNet

Solution: Change the network so learning identity functions with extra layers is easy

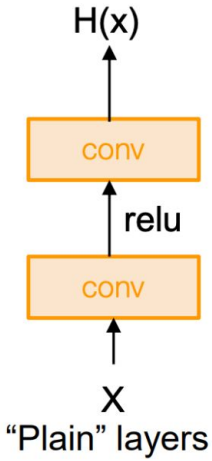


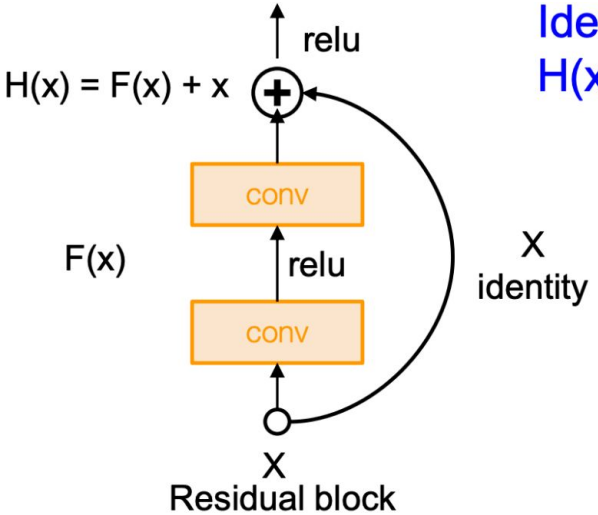
Slide adopted from: Fei Fei Li, CS231n: Deep Learning for Computer Vision, Stanford University, https://cs231n.stanford.edu/

Convolutional Neural Network

ResNet

Solution: Change the network so learning identity functions with extra layers is easy





Identity mapping:
$H(x) = x$ if $F(x) = 0$

Use layers to fit **residual** $F(x) = H(x) - x$ instead of $H(x)$ directly

Slide adopted from: Fei Fei Li, CS231n: Deep Learning for Computer Vision, Stanford University, https://cs231n.stanford.edu/

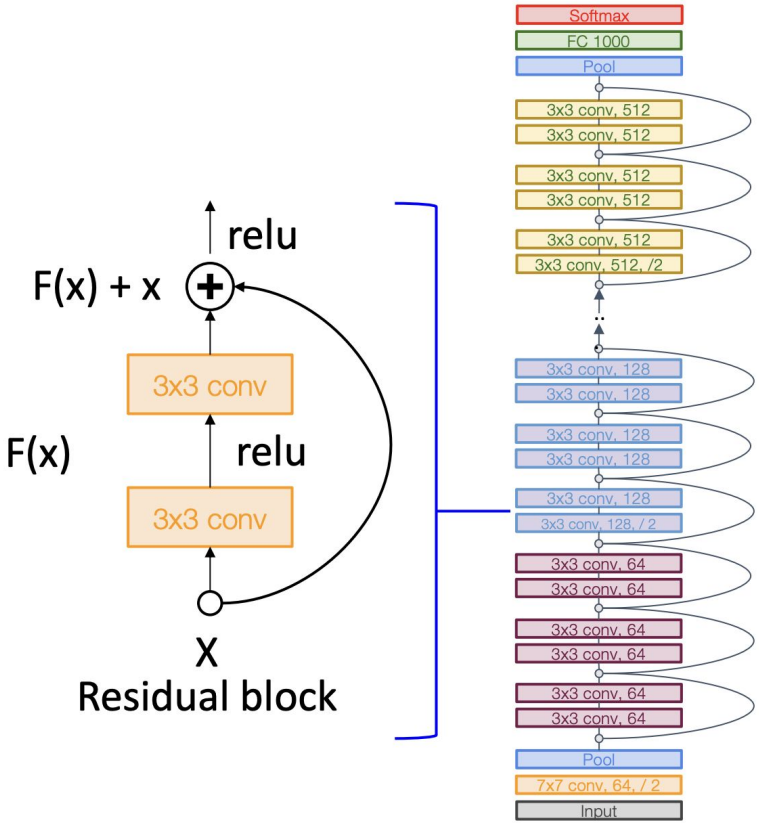
Convolutional Neural Network

ResNet

A residual network is a stack of many residual blocks

Regular design, like VGG: each residual block has two 3x3 conv

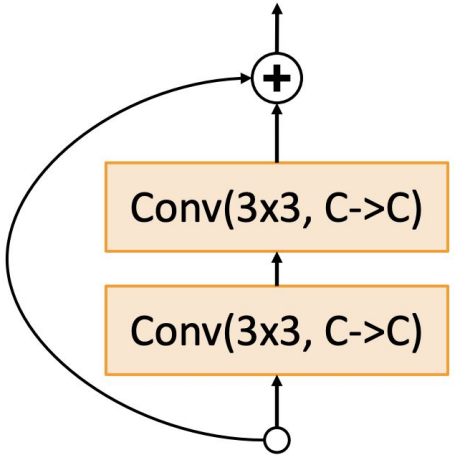
Network is divided into **stages**: the first block of each stage halves the resolution (with stride-2 conv) and doubles the number of channels



Slide adopted from: Fei Fei Li, CS231n: Deep Learning for Computer Vision, Stanford University, https://cs231n.stanford.edu/

Convolutional Neural Network

ResNet : Basic Block



FLOPs: $9HWC^2$

FLOPs: $9HWC^2$

"Basic"
Residual block

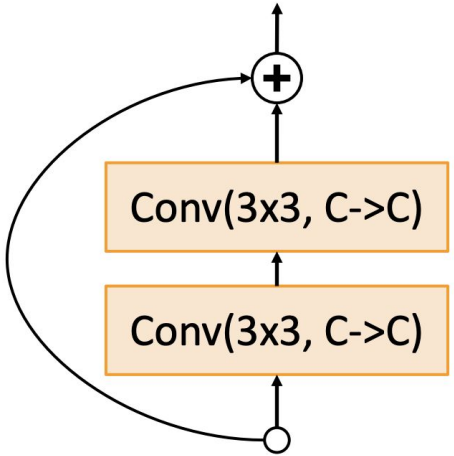
Total FLOPs:
$18HWC^2$

Slide adopted from: Fei Fei Li, CS231n: Deep Learning for Computer Vision, Stanford University, https://cs231n.stanford.edu/

Convolutional Neural Network

ResNet : Bottleneck Block

More layers, less computational cost!

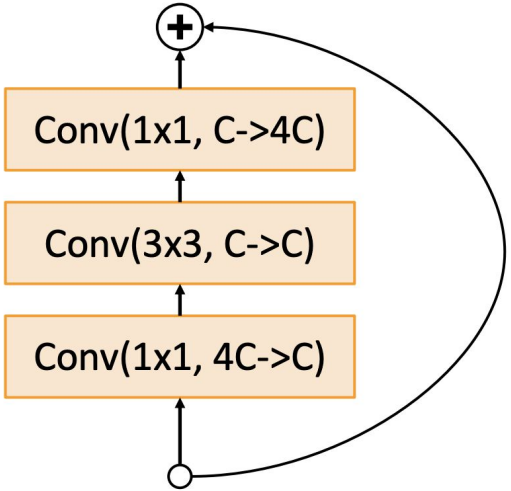


FLOPs: $9HWC^2$

FLOPs: $9HWC^2$

"Basic" Residual block

Total FLOPs: $18HWC^2$



FLOPs: $4HWC^2$

FLOPs: $9HWC^2$

FLOPs: $4HWC^2$

Total FLOPs: $17HWC^2$

"Bottleneck" Residual block

Slide adopted from: Fei Fei Li, CS231n: Deep Learning for Computer Vision, Stanford University, https://cs231n.stanford.edu/