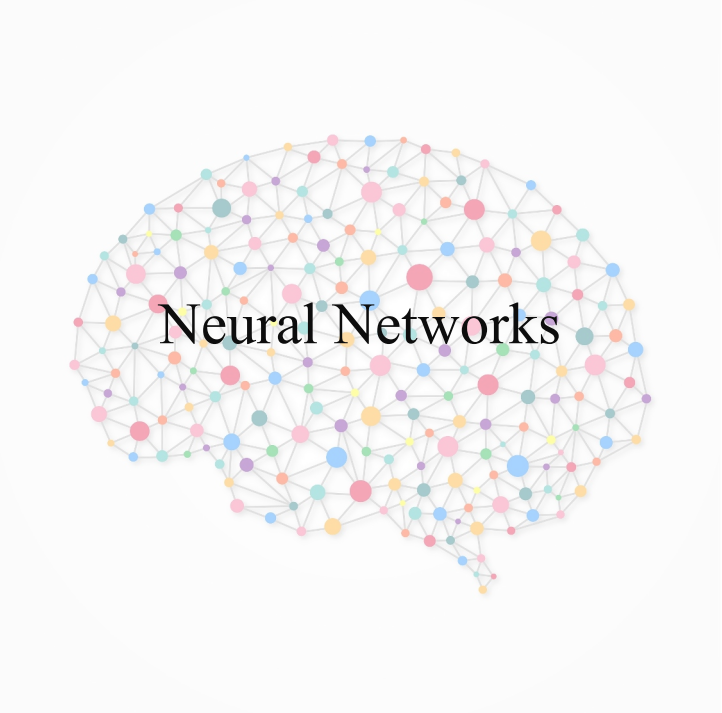
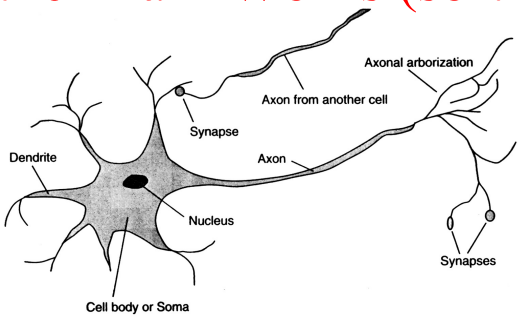


1

How the Brain Works (sort of…)



- Neuron is fundamental functional unit
  - Soma: cell body
  - Axon: long single fiber that connects to other neurons
  - Dendrites: connected to axons from other neurons
  - Synapse: connecting junction
- A collection of simple cells can lead to thought, action, and consciousness ("bottom-up" statement)

2

2

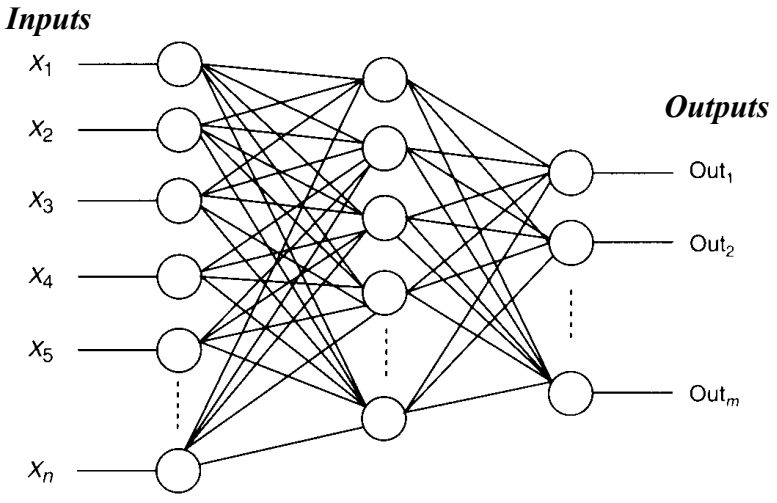
Network Structures

- Two main varieties
  - Feed-forward
    - Unidirectional links with no cycles
    - Directed acyclic graph (DAG)
    - Not like the brain!
      - We have memory
      - Many back connections
  - Recurrent
    - Links can form arbitrary topologies

3

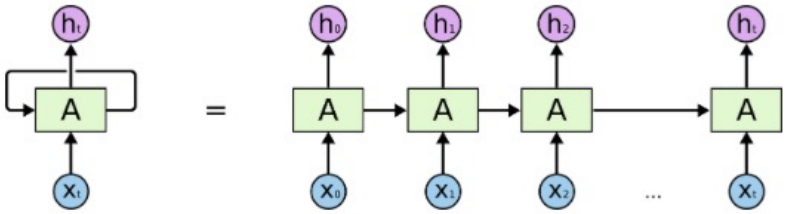
3

Feed-Forward Neural Network



4

Recurrent Neural Network



5

5

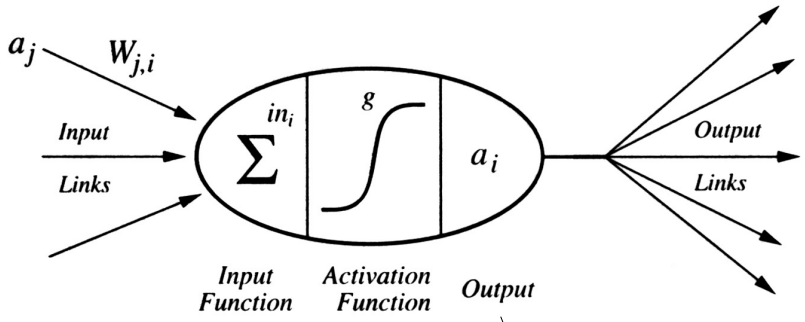
Neural Networks

- Neural net is composed of nodes (units)
  - Some connected to outside world as input or output units
- Nodes are connected by links
  - Input and output links
- Each link has numeric weight associated with it
  - Primary means of **long-term storage/memory**
  - Weights are modified to bring network's input/output behavior to goal response
- Nodes have activation level
  - Given its inputs and weights
  - Local computation based on inputs from neighbors (no global control)

6

6

Node Computation



$$\sum_j W_{j,i} a_j = \boldsymbol{W} \cdot \boldsymbol{a}$$ linear

$$g(\sum_j W_{j,i} a_j)$$ non-linear

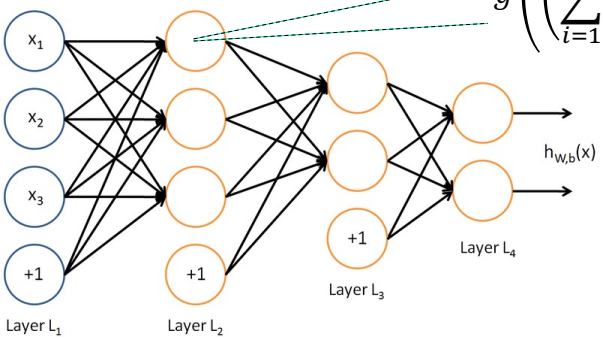
7

7

Additive Bias Terms

linear term bias

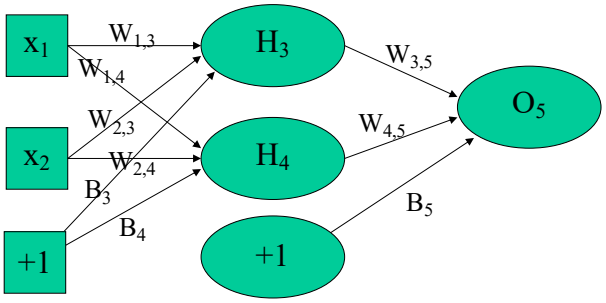
$$g\left(\left(\sum_{i=1}^{3} W_{i,1} \cdot x_i\right) + \left(B_{L_1} \cdot 1\right)\right)$$



Common to have a "bias" term <u>for each node</u> to "*shift*" linear inputs into good place for activation function (learn best bias values)

8

Simple Feed-Forward Neural Network



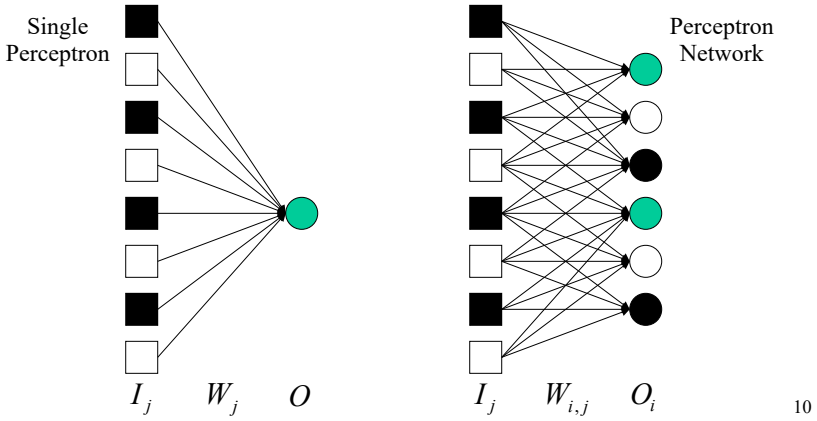
$$O_5 = g(\ W_{3,5} \cdot g(W_{1,3} \cdot x_1 + W_{2,3} \cdot x_2 + B_3) + W_{4,5} \cdot g(W_{1,4} \cdot x_1 + W_{2,4} \cdot x_2 + B_4) + B_5\ )$$

Learning just becomes a process of tuning parameters to fit data in training set!!!

9

"Perceptrons"

- First studied in late 1950's
- Single-layer, feed-forward network

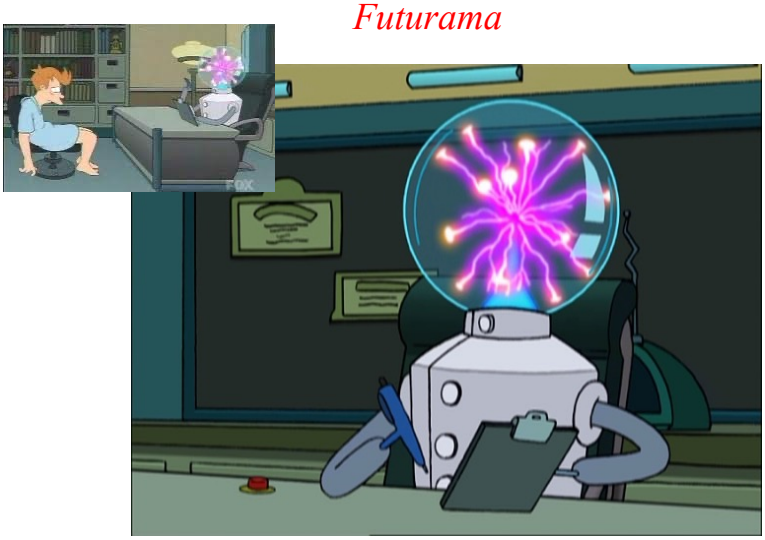


10

10

Dr. Perceptron

Futurama



11

11

Perceptrons

- Consider simple "*Threshold*" (sign) activation function:

$$g\left(\sum_j W_{j,i}a_j\right) = g\left(\langle W_{1,i}, W_{2,i}, W_{3,i}\rangle \cdot \langle a_1, a_2, a_3\rangle\right)$$

where $g = 1$ if $\langle W_{1,i}, W_{2,i}, W_{3,i}\rangle \cdot \langle a_1, a_2, a_3\rangle > 0$

$g = -1$ if $\langle W_{1,i}, W_{2,i}, W_{3,i}\rangle \cdot \langle a_1, a_2, a_3\rangle \leq 0$

- Now consider equation for a line

$$y = mx + b$$
$$0 = mx + b - y$$
$$0 = \langle m, b, -1\rangle \cdot \langle x, 1, y\rangle$$

$$0 = \langle W_{1,i}, W_{2,i}, W_{3,i}\rangle \cdot \langle a_1, a_2, a_3\rangle$$

12

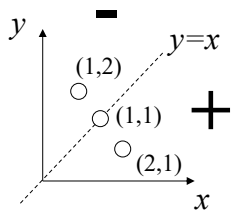
12

Dividing the Space (+,-)

$y = x \longrightarrow 0 = x - y \longrightarrow 0 = \langle 1, -1 \rangle \langle x, y \rangle$

(1,2):

-1 = 1 - 2



(2,1):

+1 = 2 - 1

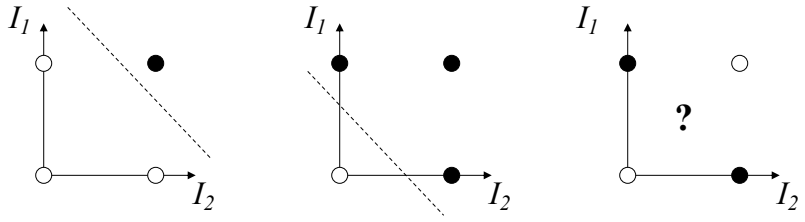
Threshold perceptrons represent functions that are **linearly separable** ***

13

13

Linear Separability in Perceptrons

Limited in Boolean functions they can represent
AND, OR, but not XOR
(True = solid circle)



I_1 AND I_2 I_1 OR I_2 I_1 XOR I_2

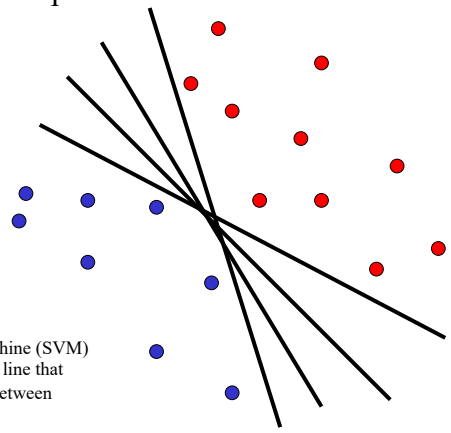
"A perceptron can represent a function only if some line can separate all white dots from black dots"

14

14

Linear Classifiers

- Multiple Perceptron solutions to separate positive and negative examples



Note: Support Vector Machine (SVM) approach seeks to find the line that maximizes the "margin" between classes – better!

15

Perceptron Learning Algorithm

- Initially assign random weights
  - More on this later…
- Update network to try to make consistent with examples
  - Make small adjustments in weights to reduce difference between observed and predicted values
  - Updating process divided into "epochs"
    - One epoch involves updating all weights for all examples

16

16

Error ("Loss") Function

Squared-Error function (**to be summed over <u>all</u> training examples – here only shown for 1 example**): [Note: there are <u>multiple</u> types of error/loss functions]

$$E = \frac{1}{2} Err^2 = \frac{1}{2}\left[O - g\left(\sum_j W_j a_j \right) \right]^2$$

Desired output value

Perceptron output value

Find the weights W_j that minimize this error (for all training examples)

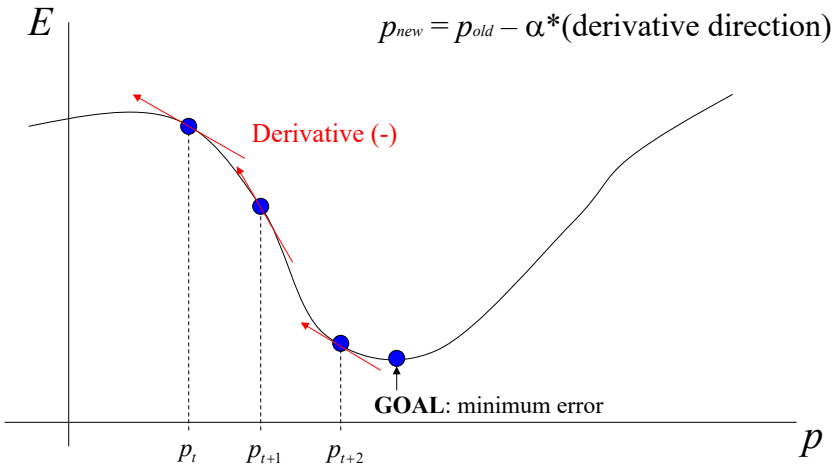
How determine these weight values????

Gradient Descent method

17

17

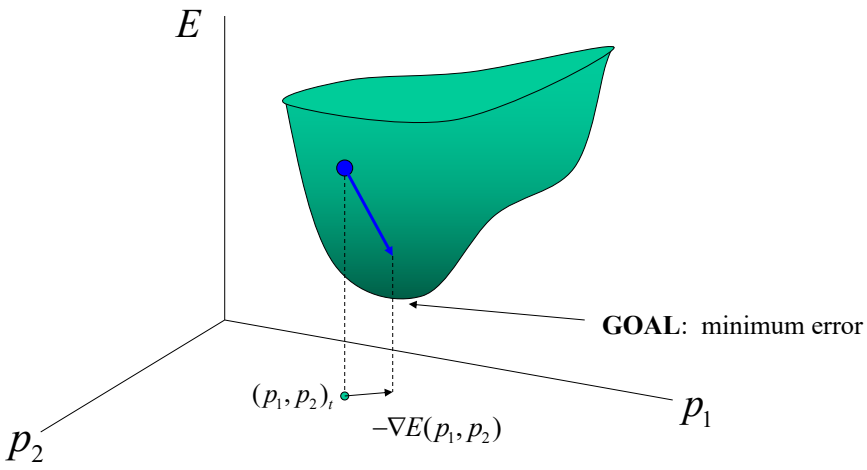
Gradient Descent Visualization (1-D)



18

18

2-D Visualization



19

19

Weight Updating via Gradient Descent

Squared-Error function (**again, to be summed over <u>all</u> training examples**):

$$E = \frac{1}{2} Err^2 = \frac{1}{2}\left[O - g\left(\sum_j W_j a_j \right) \right]^2$$

Desired output value

Perceptron output value

$$\begin{aligned} \frac{\partial E}{\partial W_j} &= \frac{1}{2}\frac{\partial Err^2}{\partial W_j} = Err \cdot \frac{\partial Err}{\partial W_j} \\ &= Err \cdot \frac{\partial}{\partial W_j}\left[O - g\left(\sum_j W_j a_j \right) \right] \\ &= Err \cdot g'() \cdot (-a_j) \end{aligned}$$

Note: $g'()$ is omitted from "threshold" perceptrons

$$\begin{aligned} W_j &= W_j - \alpha \cdot \frac{\partial E}{\partial W_j} \\ &= W_j + \alpha \cdot Err \cdot g'() \cdot a_j \end{aligned}$$

Note: Save the <u>new</u> weight values and do all weight updates **together** <u>at the end</u> of the epoch

20

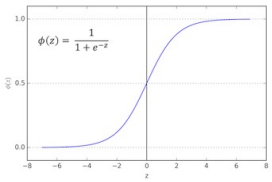
20

Common Activation Functions

Sigmoid

$$Sig(x) = \frac{1}{1 + e^{-x}}$$

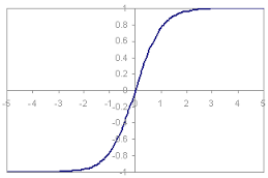
0 to 1



TANH

$$tanh(x) = 2sig(2x) - 1$$

-1 to +1



21

Issues

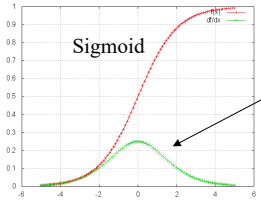
- Sigmoid can have "saturation" problems when input is composed of high (+/-) weights

$$output = \frac{1}{1 + e^{-input}}$$

$(input = x_1 * w_1)$

If $|w_1|$ is large, then *output* is near 0 or 1

$$\frac{\partial out}{\partial net} = out * (1 - out)$$



Thus the gradient will be near zero… **Bad for learning** with Gradient Descent (slows down or makes no changes)!

22

22

"Vanishing Gradient" Problem

- In some cases, some neurons can even "die" in the training and become ineffective
- Gradient signals from the error function decrease exponentially as they are backpropogated to earlier layers (more on this later…)
  - Derivative of certain activation functions (sigmoid, tanh) is small (<1), which when stacked in network layers get smaller, and smaller
- Can be problematic for deep networks!

23

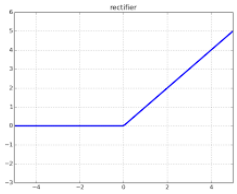
23

Common Activation Functions

Rectified Linear Unit (ReLU)

$\text{ReLU}(x) = \max(0, x)$

0 to Inf



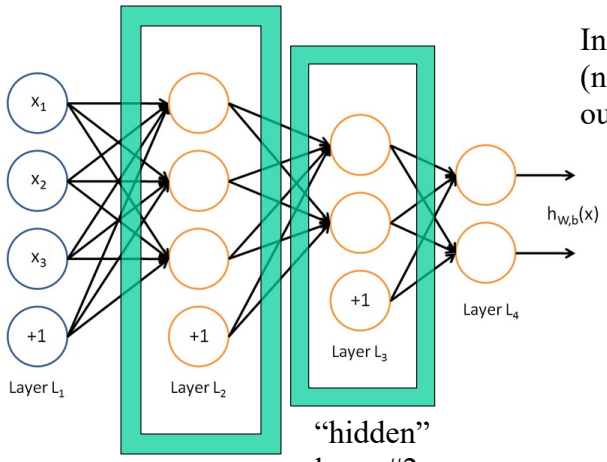
- Pros:
  - Non-saturating (like sigmoid)
    - ReLU overcomes this particular issue (derivative=1 when signal >0)
  - Faster convergence
  - Efficient computation
  - Resulting representation is "sparse" (not all having nodes have non-zero activations), removing redundancies
- "Leaky" ReLU
  - Use small negative slope around 0

ReLU works as a detector, with a particular signal being detected when the corresponding input (x) is large enough (greater than zero). In practice, the inputs are usually centered (zero-mean) and therefore a threshold of zero is reasonable. But generally use an additive bias term/value to enable a "shift" if needed.

24

24

How Handle **Multilayer** Feed-Forward Networks?



Includes "hidden" layers (not containing input or output nodes)

"hidden" layer #1

"hidden" layer #2

25

25

Learning/Training in Multilayer Feed-Forward Networks

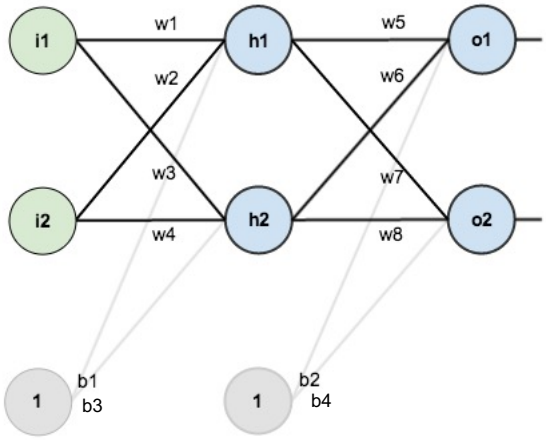
- **Back-propagation** learning algorithm
  - Divide error "locally" among contributing weights and update layer-by-layer backwards
- Example

  https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

26

26

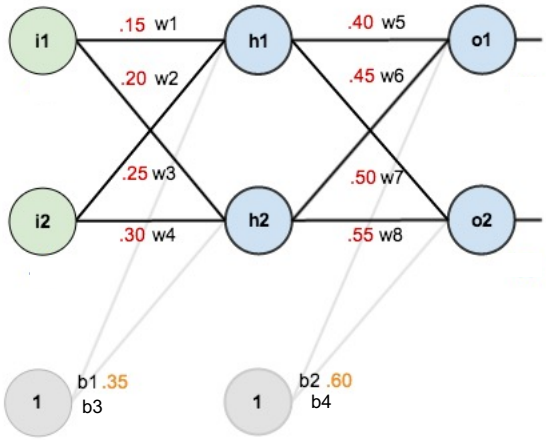
Example Network w/ 1 Hidden Layer



27

27

Initialize Weights/Biases

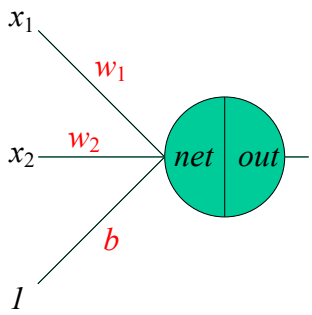


Note: common to start with zero bias value

28

28

Node Computations: "net" and "out"



Sigmoid activation:

$$net = x_1 * w_1 + x_2 * w_2 + b$$

linear

$$out = \frac{1}{1 + e^{-net}}$$

non-linear

29

29

"Forward Pass"

Note: updating with only 1 training example

- **Input (2-D):** i1=.05, i2= 0.1
  - Desired "target" output (o1=0.01, o2=0.99)
- **Output (2-D):** out_{o1}=0.7569, out_{o2}=0.7677
- **Error/Loss:**

$$E_{total} = \sum \frac{1}{2}(target - output)^2 = E_{o1} + E_{o2} = 0.3037$$

$$E_{o1} = \frac{1}{2}(0.01 - 0.7569)^2 = 0.2790$$

$$E_{o2} = \frac{1}{2}(0.99 - 0.7677)^2 = 0.0247$$

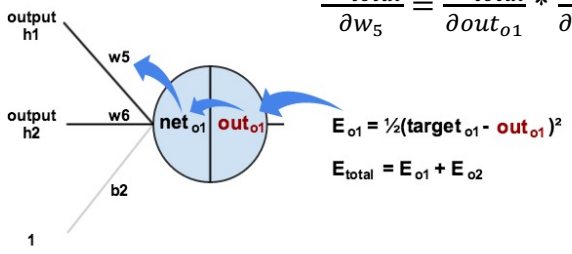
30

30

"Backward Pass"

- At final Output Layer (initially)
  - Want to know how much a change in each of w_5, w_6, w_7, w_8, b_2 , b_4 affect Total Error (for gradient descent)
  - So for w_5, we want to know $\frac{\partial E_{total}}{\partial w_5}$
  - Using the chain rule!!!

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$



31

31

Partial Derivative Terms

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

$$E_{total} = \frac{1}{2}(target_{o1} - out_{o1})^2 + \frac{1}{2}(target_{o2} - out_{o2})^2$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = -(target_{o1} - out_{o1})$$

$$= -(0.01 - 0.7569) = \mathbf{0.7469}$$

32

32

Partial Derivative Terms

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \boxed{\frac{\partial out_{o1}}{\partial net_{o1}}} * \frac{\partial net_{o1}}{\partial w_5}$$

$$out_{o1} = \frac{1}{1 + e^{-net_{o1}}}$$

$$\frac{\partial out_{o1}}{\partial net_{o1}} = out_{o1} * (1 - out_{o1}) = \mathbf{0.1840}$$

33

33

Partial Derivative Terms

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \boxed{\frac{\partial net_{o1}}{\partial w_5}}$$

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2$$

$$\frac{\partial net_{o1}}{\partial w_5} = out_{h1} = \mathbf{0.5945}$$

Note: Uses current inputs and weights to compute sigmoid response at node $h1$ (know this value **from forward pass**)

34

34

Putting it Together…

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

$$\frac{\partial E_{total}}{\partial w_5} = 0.7469 * 0.1840 * 0.5945$$
$$= \mathbf{0.0817}$$

Alternatively see written as "delta rule":

$$\frac{\partial E_{total}}{\partial w_5} = \partial_{o1} * \frac{\partial net_{o1}}{\partial w_5} = \partial_{o1} * out_{h1}$$

where

$$\partial_{o1} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} = \frac{\partial E_{total}}{\partial net_{o1}}$$

35

35

Updating via Gradient Descent

$$w_5 = w_5 - \alpha \cdot \frac{\partial E_{total}}{\partial w_5}$$

$$w_5 = 0.4 - \underset{\textit{given}}{(0.5)} \cdot 0.0817 = 0.3592$$

Repeat entire process for w_6, w_7, w_8, b_2, and b_4

Note: Save these new weight values and do all weight updates together at the end of the epoch (after all new weight values are collected, in this example)

36

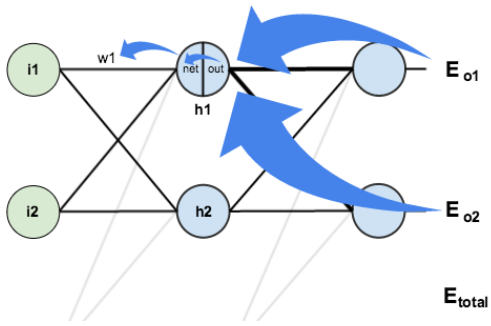
36

Now Update for "Hidden Layer(s)"

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$

"This is a bit more tricky!"



$E_{total} = E_{o1} + E_{o2}$

37

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$

$$\frac{\partial E_{o1}}{\partial out_{h1}} = \underbrace{\frac{\partial E_{o1}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}}}_{\text{Have already for } w_5 \text{ ("delta" term)}} * \frac{\partial net_{o1}}{\partial out_{h1}}$$

$$\frac{\partial net_{o1}}{\partial out_{h1}} = w_5 = 0.4$$

Note: $net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2$

Note: $\frac{\partial E_{o1}}{\partial out_{o1}}$ same as $\frac{\partial E_{total}}{\partial out_{o1}}$

$$= \underbrace{(0.7469 * 0.1840)}_{\text{Already have these values from layer just analyzed}} * 0.4 = \mathbf{0.0550}$$

38

38

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$

$$\frac{\partial E_{o2}}{\partial out_{h1}} = \underbrace{\frac{\partial E_{o2}}{\partial out_{o2}} * \frac{\partial out_{o2}}{\partial net_{o2}}} * \frac{\partial net_{o2}}{\partial out_{h1}}$$

Have already from updating w_7 ("delta" term)

$$\frac{\partial net_{o2}}{\partial out_{h1}} = w_7 = 0.5$$

Note: $net_{o2} = w_7 * out_{h1} + w_8 * out_{h2} + b_4$

$$= (\mathbf{-0.2223 * 0.1783}) * \mathbf{0.5} = \mathbf{-0.0198}$$

39

39

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = 0.0550 - 0.0198 = \mathbf{0.0352}$$

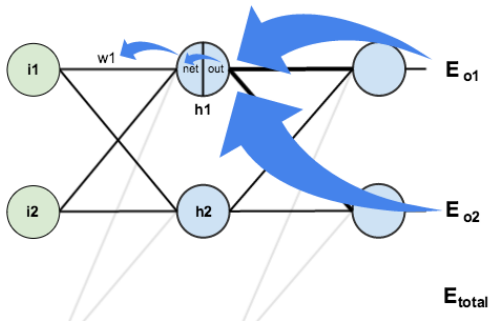
40

40

Now Update for "Hidden Layer(s)"

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial out_{h1}}{\partial net_{h1}} = out_{h1} * (1 - out_{h1}) = \mathbf{0.2411}$$



$E_{total} = E_{o1} + E_{o2}$

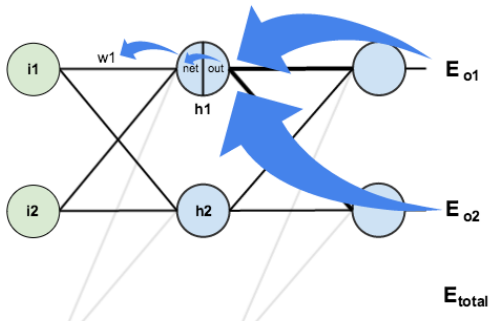
41

Now Update for "Hidden Layer(s)"

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial net_{h1}}{\partial w_1} = i_1 = \mathbf{0.05}$$

$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1$$



$E_{total} = E_{o1} + E_{o2}$

42

Putting it Together…

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial w_1} = 0.0352 * 0.2411 * 0.05$$
$$= \mathbf{0.0004243}$$

43

43

Updating via Gradient Descent

$$w_1 = w_1 - \alpha \cdot \frac{\partial E_{total}}{\partial w_1}$$

$$w_1 = 0.15 - (0.5) \cdot 0.0004243 = 0.1498$$

Repeat entire process for w_2, w_3, w_4, b_1, and b_3

Note: Save these new weight values and do all weight updates together at the end of the epoch

44

44

More on Loss Functions…

45

45

SoftMax Output

- If target output is a vector of **dependent multi-class membership**, it can be desirable to train the network to output a "*probability distribution*" across the classes
  - Multinomial "hard targeting" [1 0 0], [0 1 0], [0 0 1]
    - Also called "**1-hot**" vectors, used as **ground-truth targets**
  - Network outputs: [.99 0 .01], [.15 .85 0], [.2 .1 .7]
- Use normalized exponential of outputs out_{oi}
  - Sums to 1 (probability)
  - "Peaky" distribution

$$p_{oi} = \frac{e^{out_{oi}}}{\sum_j e^{out_{oj}}}$$

46

46

Multi-Class "*Categorical*" **Cross Entropy** Error/Loss Function

- Softmax output: $p_{oi} = \frac{e^{out_{oi}}}{\sum_j e^{out_{oj}}}$

$$E = -\sum_i target_{oi} * \ln(p_{oi})$$

Note: only one $target_{oi}$ is 1 (rest are 0)

$$\frac{\partial E}{\partial out_{oi}} = (p_{oi} - target_{oi})$$

Used in Gradient Descent

47

47

Calibration and Temperature Scaling

- Would like to have Softmax be representative of true posterior probability
  - If largest softmax output of a class is .8, then want that to mean "*there is an 80% chance of this classification being correct*" [this is perfect calibration]
- There exist methods to assess the calibration quality
  - "On Calibration of Modern Neural Networks"
  - https://arxiv.org/pdf/1706.04599.pdf
- If uncalibrated (typical), can attempt to calibrate model
  - Temperature (T) scaling
  - Either soften or sharpen softmax vals
  - Can learn or search for best T

$$p_{oi} = \frac{e^{out_{oi}/T}}{\sum_j e^{out_{oj}/T}}$$

48

48

Training Particulars

49

49

Data Preprocessing

- Mean subtraction
  - Subtract the mean across every individual *feature/dimension* in the data
  - Method 1: Compute from dataset
  - Method 2: From fixed value of data range (e.g., 128 for 8-bit values)
- Normalization (after mean subtraction)
  - Scale the data so they have same range
    - Method 1: Divide each dimension by its standard deviation
    - Method 2: Normalize each dimension so that the *min* and *max* values along the dimension is -1 and 1
    - Method 3: Divide by shifted data range extent (e.g., $128)_{50}$

50

Weight Initialization

(for "link" weights, biases are set to 0)

- Method 1a: Gaussian
  - Sample from Gaussian with $\sigma^2=1$, then scale
    - Scale with a small value (e.g., .001)
- Method 1b: Gaussian (Xavier/Glorot)
  - For the set of weights that go in/out of a node (N_{in}, N_{out}), sample from Gaussian with $\sigma^2 = 2 / (N_{in}+N_{out})$
  - Not designed for ReLU activations, better with Sigmoid
- Method 1c: Gaussian (Kaiming/He)
  - Use $\sigma^2 = 2 / N_{in}$ or $\sigma^2 = 2 / N_{out}$
  - Designed for ReLU
- Others…

51

51

Random Seed Initialization

- Training is highly affected by the initial **random seed** used to set the initial network weights and selection of data order
  - Can give large difference in results with different seed
- Can training multiple times, each time with a different random seed initialization, and choose final model giving best results
  - Could also make into an ensemble of models
- Should report accuracy **mean, stdev** for multiple trained models to truly understand performance (not just one run/model)
  - Use statistical test to prove one method is better than another

"Revisiting Batch Norm Initialization"
J. Davis and L. Frank
European Conference on Computer Vision, October 2022

52

52

Learning Rate α

$$w_5 = w_5 - \boxed{\alpha \cdot} \frac{\partial E_{total}}{\partial w_5}$$

How to choose/set learning rate α?

53

53

"Step" Reduction

- Begin with a selected starting learning rate

  $\alpha = 0.1$

- At pre-selected epochs intervals (e.g., 30th, 60th, 90th, …), divide current learning rate by 10

  $\alpha^{new} = \alpha^{old} / 10$

54

54

"Poly" Learning Rate Adaptation

- Begin with a selected starting learning rate

$$\alpha = 0.1$$

- The learning rate is multiplied by a changing rate (decay) at each iteration (e.g., epoch)

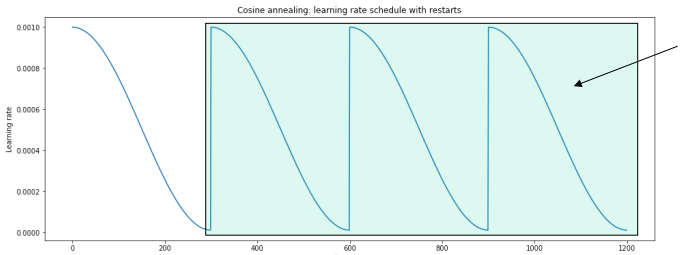
$$\alpha^{new} = \alpha^{old} * \left(1 - \frac{iter}{max\ iter}\right)^{power}$$

$power = 0.9$

55

55

Cosine Learning Rate



Note: can do only 1 reduction w/ no cycles

$$\eta_t = \eta^i_{\min} + \frac{1}{2}\left(\eta^i_{\max} - \eta^i_{\min}\right)\left(1 + \cos\left(\frac{T_{current}}{T_i}\pi\right)\right)$$

where η_t is the learning rate at timestep t (incremented each mini batch), $\eta^i_{\max}$ and $\eta^i_{\min}$ define the range of desired learning rates, $T_{current}$ represents the number of epochs since the last restart (this value is calculated at every iteration and thus can take on fractional values), and T_i defines the number of epochs in an cycle.

"By drastically increasing the learning rate at each restart, we can essentially exit a local minima and continue exploring the loss landscape."

56

56

"Stochastic" Gradient Descent

- Perform a parameter update for **each** training data example only (instead of all data)
  - Usually much faster, can learn online

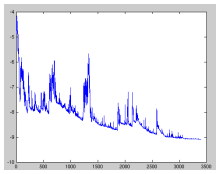
while not converged
  shuffle/randomize data examples x
  for **each** training example x_j
    for each parameter $p_i \in p$
      compute gradient (using only x_j) $\nabla_{pi}E(p, x_j)$
      update p_i with $p_i = p_i - \alpha * \nabla_{pi}E(p, x_j)$

***** Note: do simultaneous update of all parameters p_i at end of each iteration (after each training example)!**

57

57

Stochastic Gradient Descent



- Performance
  - Fluctuates
    - Can jump to better local minima
    - However can keep overshooting (better if slowly decrease learning rate)
- **"Mini-Batch" Gradient Descent**
  - Perform an update using mini-batch of **N data examples** (e.g., 32-256 examples instead of just 1)
  - Can have **more stable/smoother convergence**, more efficient
  - Use batch sampler to get appropriate batches during training
    - Configure to get equal % of examples from each class when have unbalanced training classes

58

58

Momentum

- Can oscillate across slopes of ravine while only making small progress along weaker bottom slope towards local minima
- Solution: Remember the update at each iteration, and determine next update as combination of current gradient and previous update:

Old: $p_i = p_i - \alpha * \nabla E(p_i)$

$$\Delta p_i = (\alpha * \nabla E(p_i) + \beta * \Delta p_i)$$

current remember last time

update p_i with $p_i = p_i - \Delta p_i$

(typically start with $\beta = .5$ then increase to $\beta = .9$)

New: $p_i = p_i - (\alpha * \nabla E(p_i) + \beta * \Delta p_i)$

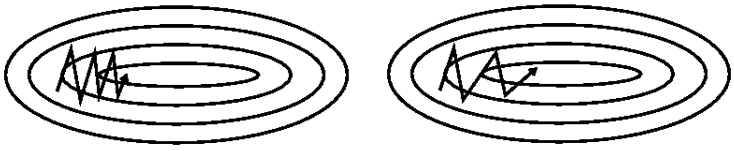
- Result: Tends to keep traveling in same direction, preventing oscillations

59

59

Momentum

- Analogy: push a ball down a hill. It accumulates momentum as it rolls downhill, becoming faster and faster on the way
- Hence, momentum term increases for dimensions whose gradients point in same directions and reduces updates for dimensions whose gradients change direction (cancel out)
  - Gain faster convergence and reduced oscillation



w/o momentum w/ momentum

60

60

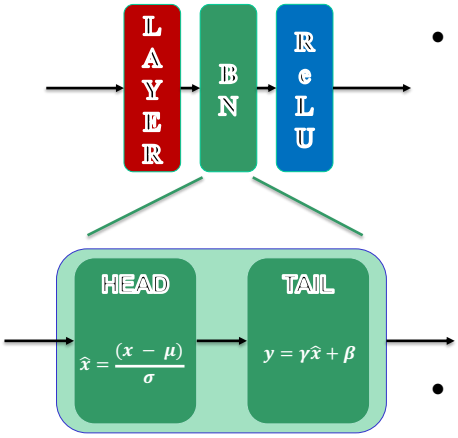
Batch Normalization (BN)

- Technique to provide any (each) layer with inputs that are zero mean and unit variance
  - Also has learnable parameters of scale, shift
  - Makes the data comparable across features
- Normalize for each feature going into a node using the input mini-batch examples
- Helps in two ways: faster learning and higher overall accuracy

61

61

Batch Normalization (BN)



- Frequently used in NN before activation
  - **Head**: Normalize / whiten data
    - Mean = 0
    - Variance = 1
  - **Tail**: Learnable affine transformation (scale and shift)
    - Scale initialized to 1
    - Shift initialized to 0
- Helps to stabilize data through network

62

62

Batch Normalization (BN)

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

63

63

Batch Normalization (BN)

- Need to transform the batch means and variances statistics into population statistics
  - Required for new test data (single example)
- Compute and use the average across mini-batches
- NOTE: Better initialization can yield improved performance!

"Revisiting Batch Norm Initialization"
J. Davis and L. Frank
European Conference on Computer Vision, October 2022

64

64

L2 Regularization

- Helps to reduce overfitting
- Preference to learn smaller weights w_i and spread weights as not to have only a few weights with large values and the rest with zero
  - Keeps the network from focusing on specific non-generalized items

65

65

L2 Regularization ("*Weight Decay*")

Loss = E_0 "standard loss" + E_1 "weight regularization"

Note: Typically for "link" weights, not biases or BN

$$E = E_0 + E_1 = E_0 + \frac{\lambda}{2}\sum_i w_i^2$$

In Gradient Descent:

$$w_i = w_i - \alpha\frac{\delta E}{\delta w_i} = w_i - \alpha\left(\frac{\delta E_0}{\delta w_i} + \frac{\delta E_1}{\delta w_i}\right) = w_i - \alpha\frac{\delta E_0}{\delta w_i} - \alpha\lambda w_i$$

$$w_i = (1 - \alpha\lambda)w_i - \alpha\frac{\delta E_0}{\delta w_i}$$

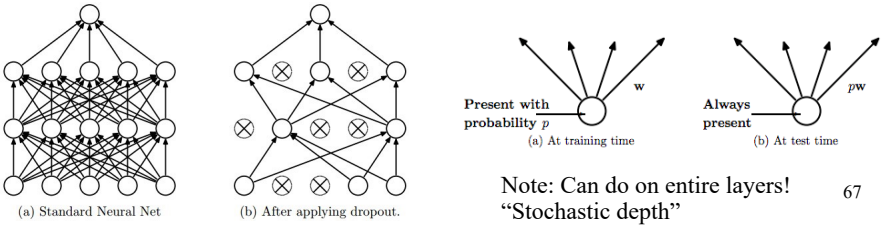
Note: Typical values of λ are 0.001 or 0.0001, but really tied to size of network!

66

66

Dropout

- Another way to prevent overfitting to training data
  - Samples a network within the full network, and only updates the parameters of the sampled network based on the input data
    - Keeps a neuron **active** with some probability (e.g., p = .5 to .9), otherwise set it to zero
    - For each training example/batch a different set of units to drop is chosen
- No dropout at test time, only during training
  - Approximate average of dropped-out networks by using the full network with each node's output weighted by p
  - Interpretation: evaluate an averaged prediction across ensemble of sub-networks



(a) Standard Neural Net (b) After applying dropout. (a) At training time (b) At test time

Note: Can do on entire layers!
"Stochastic depth"

67

67

"Deep Learning"

- Algorithms that attempt to model high-level abstractions in data by using a deep graph with multiple processing layers, composed of multiple linear and non-linear transformations
  - Exploits hierarchical explanatory factors where higher level, more abstract concepts are learned from lower level one
- Deep neural network (DNN), with "*many*" hidden layers
  - G. Hinton: **>1 hidden layer**

68

68

Convolutional Neural Networks (CNNs)

69

69

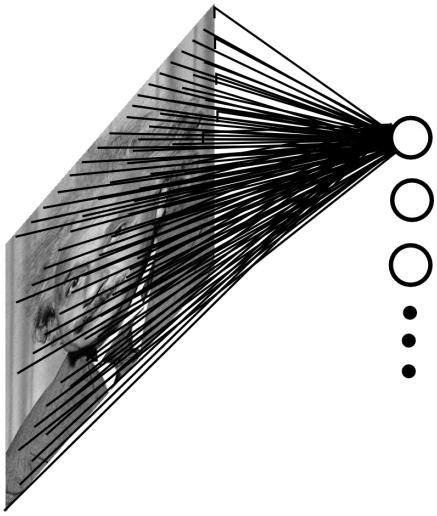
CNNs

- Type of neural network designed to take advantage of the 2D structure of "image" data
- Composed of one or more "convolutional" (with activations) and "pooling" layers, and may have fully-connected layers (as in typical artificial neural networks) at end for classification
  - Uses tied/shared weights at convolutional layers
- Can be trained with standard backpropagation

70

70

Naïve Connection

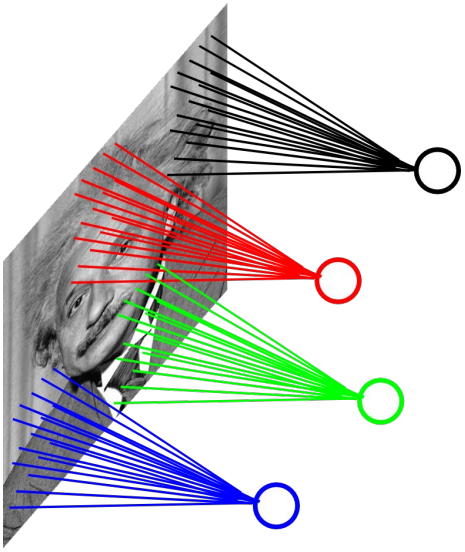


- Could connect every pixel to each node in the first hidden layer
  - Too many weights!
  - Not have enough training examples!

71

71

Locally Connected

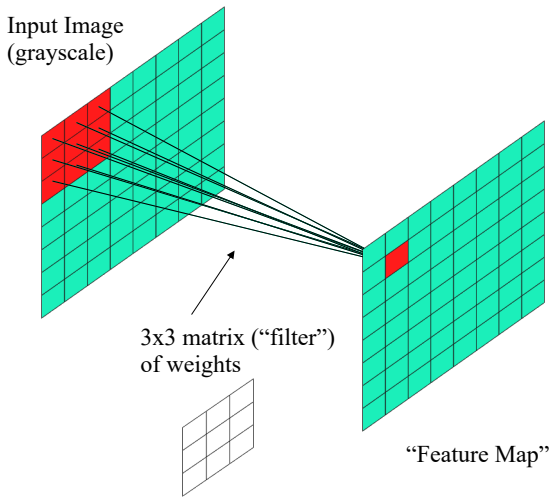


- Good when input image is registered
  - Certain visual features/patterns are in same area of the image
    - e.g., aligned "mug shot" for face recognition (nose, mouth, eyes are in same image locations)

72

72

Turn Local into "Convolution"

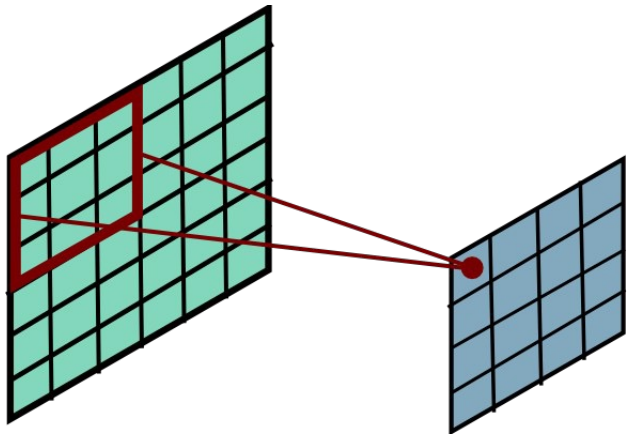


Can think of this as each location in the "Feature Map" as a node in a neural network layer, with 3x3=9 weights coming into each node. But here all of these nodes have the same "shared" weights (i.e., the feature map has a total of 9 weights associated with it to learn)

73

73

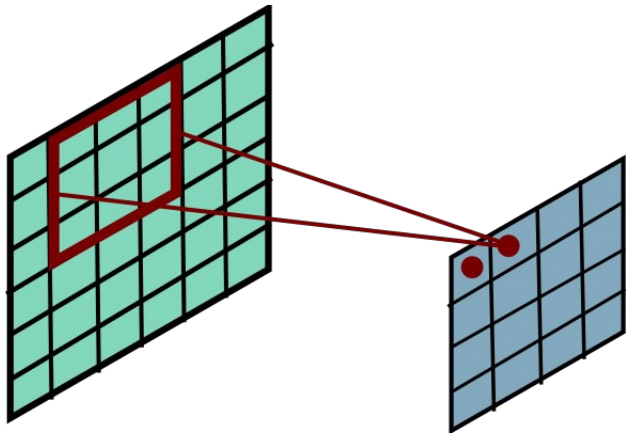
Convolution Operation (w/ 3x3 links to node)



74

74

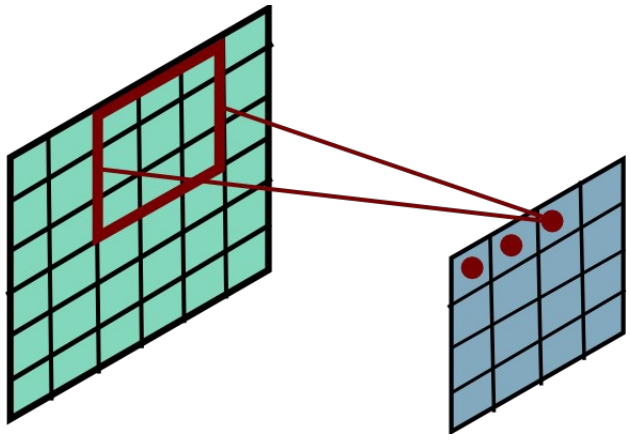
Convolution Operation
(w/ 3x3 links to node)



75

75

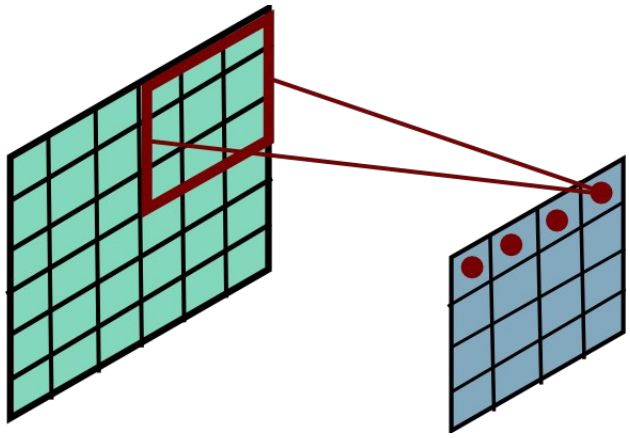
Convolution Operation
(w/ 3x3 links to node)



76

76

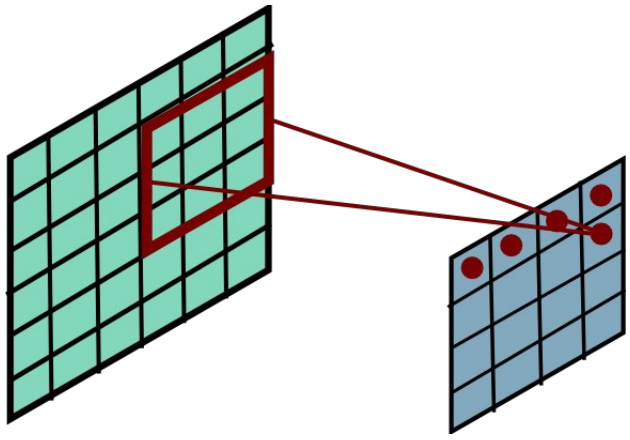
Convolution Operation
(w/ 3x3 links to node)



77

77

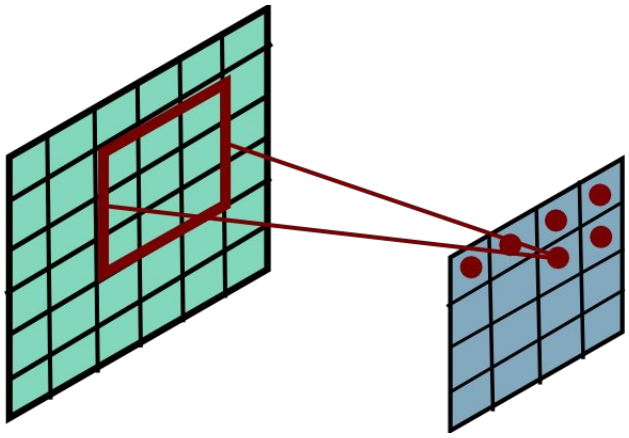
Convolution Operation
(w/ 3x3 links to node)



78

78

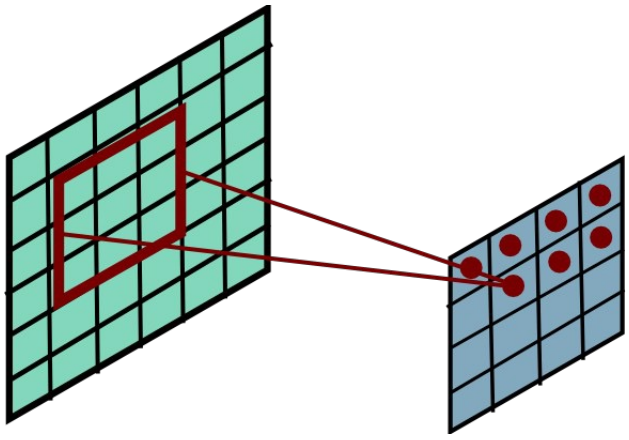
Convolution Operation
(w/ 3x3 links to node)



79

79

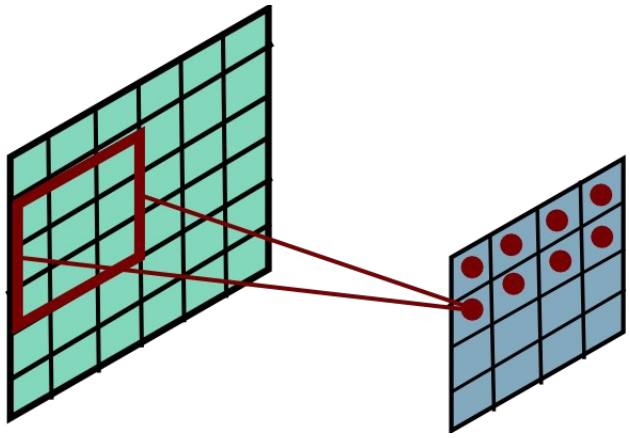
Convolution Operation
(w/ 3x3 links to node)



80

80

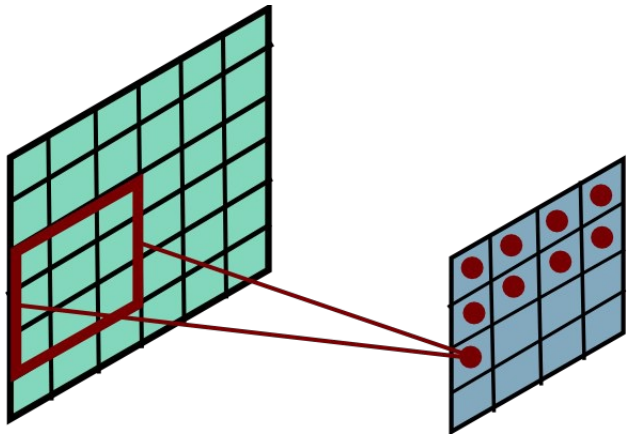
Convolution Operation
(w/ 3x3 links to node)



81

81

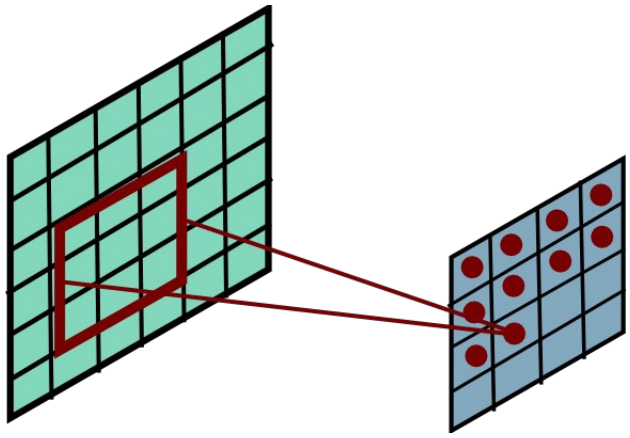
Convolution Operation
(w/ 3x3 links to node)



82

82

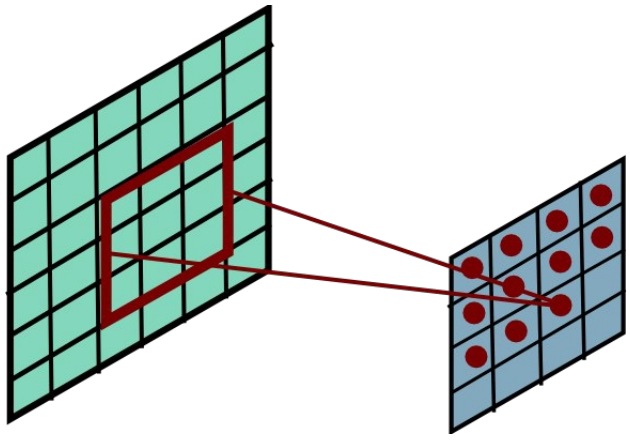
Convolution Operation
(w/ 3x3 links to node)



83

83

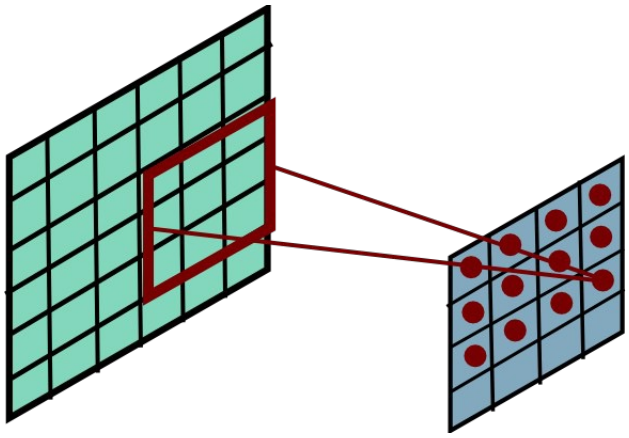
Convolution Operation
(w/ 3x3 links to node)



84

84

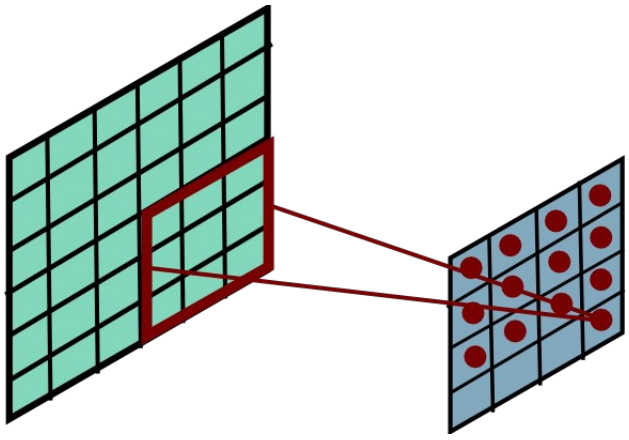
Convolution Operation
(w/ 3x3 links to node)



85

85

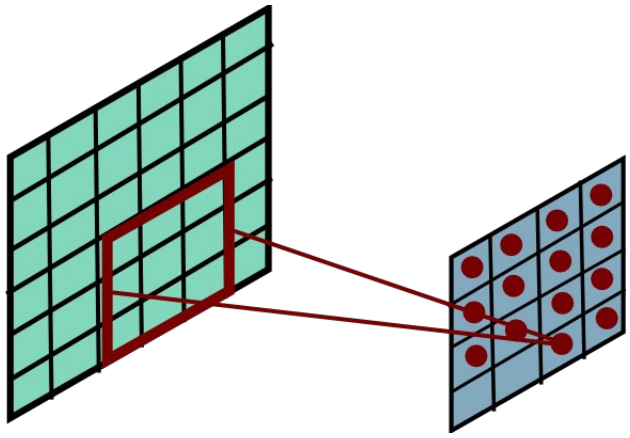
Convolution Operation
(w/ 3x3 links to node)



86

86

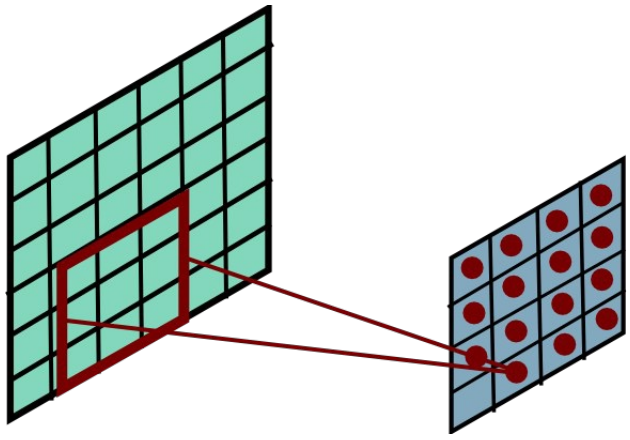
Convolution Operation
(w/ 3x3 links to node)



87

87

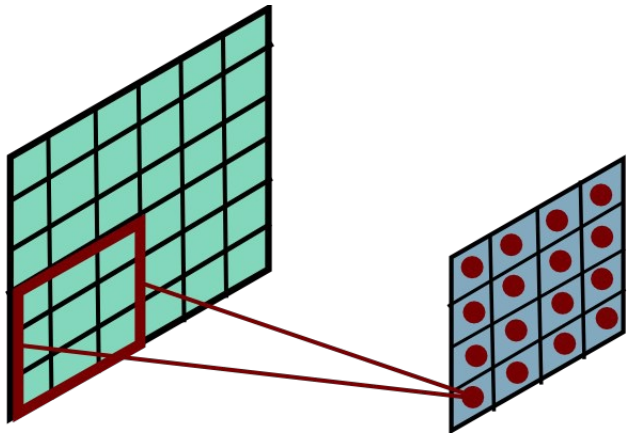
Convolution Operation
(w/ 3x3 links to node)



88

88

Convolution Operation (w/ 3x3 links to node)



89

89

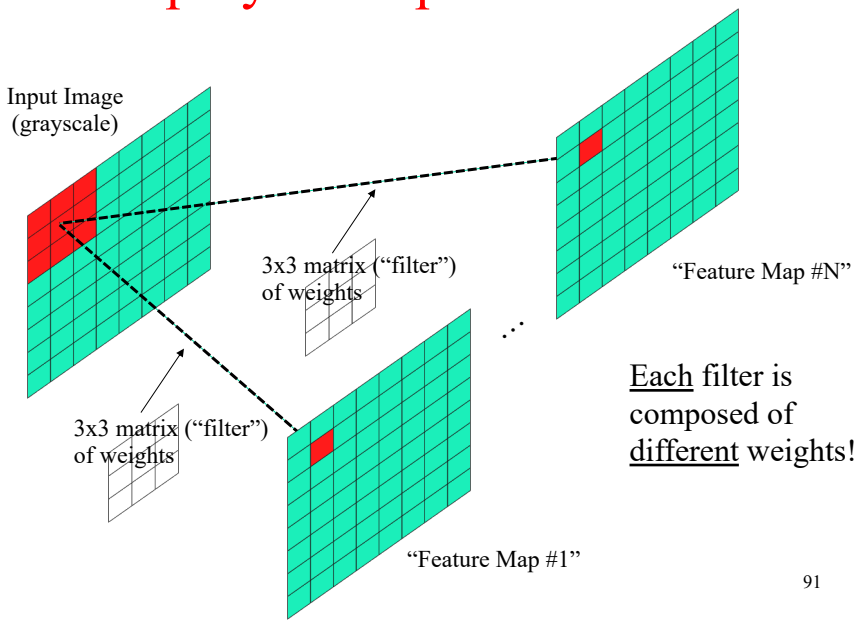
"Stride"

- "Stride" is the number of shifts in pixels over the input matrix
  - Stride = 1: moves the filter in steps of 1 pixel at a time
  - Stride = 2: moves the filter in steps of 2 pixels at a time

90

90

Employ Multiple "Filters"…



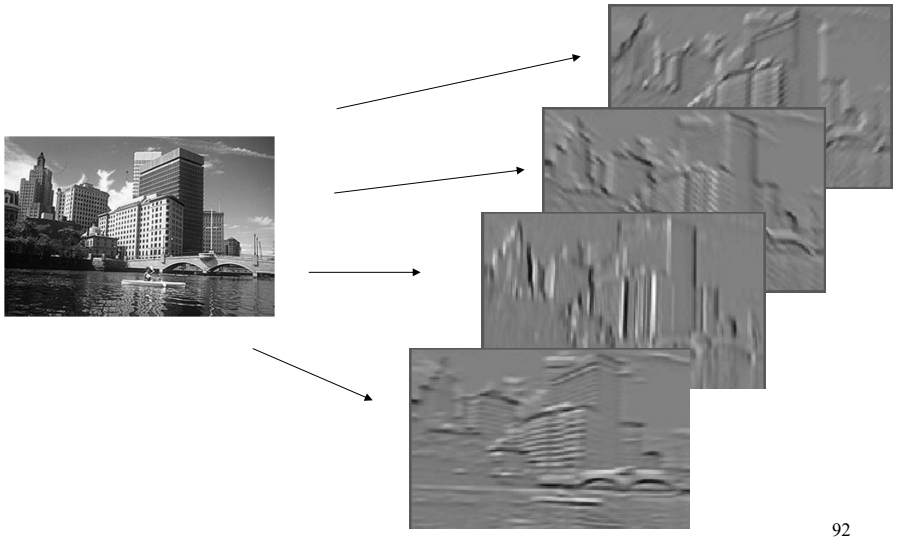
Each filter is composed of different weights!

91

91

"Convolutional Layer"

Multiple Filters to Create Feature Maps



92

92

Filter Sizes

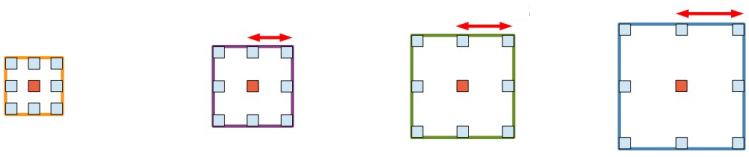
- Can choose sizes of 1x1, 3x3, 5x5, or 7x7, etc. throughout the network layers
- Some networks have larger-sized convolution filters (and possibly with larger strides) in earlier network layers
- **Training will "fill-in" (learn) the necessary filter weights/values**

93

93

Atrous Convolutions

- Convolution with upsampled filters ("Atrous convolution")
- Method to effectively enlarge the field-of-view of filters (larger context) without increasing the number of parameters or the amount of computation.



Also referred to as "dilation"

94

94

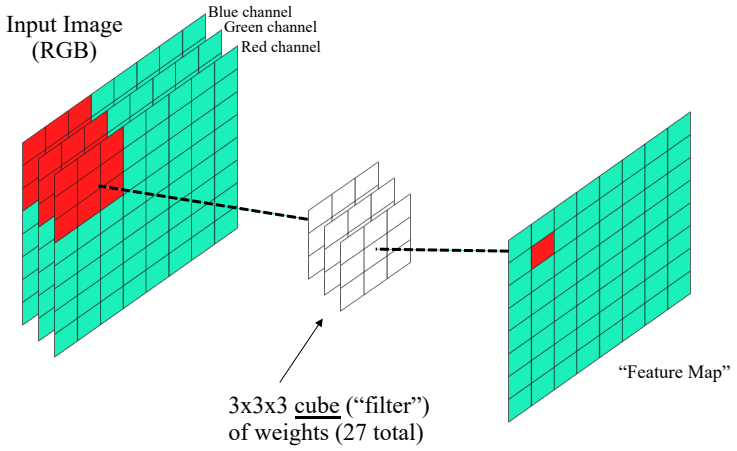
What About Input with Color (Multi-Channel) Image/Input?

- For grayscale image, may use 3x3 filter
  - Output is same row,col size as input
- As color image has 3 channels (RGB), use 3x3x3 filter (cube)
  - Output is same row,col size as input with only 1 channel (not 3)
- So if desire 64 feature maps with 3x3 spatial filter in initial conv layer for RGB input image:
  - Will have 64 filters of size 3x3x3 to learn
  - This "cube" concept extends to deeper layers…

95

95

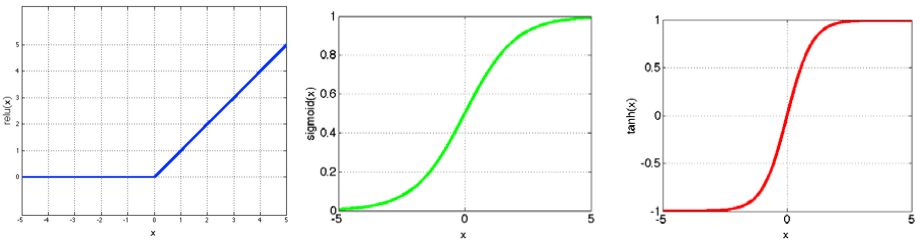
Filter "Cube"



96

Activation Function

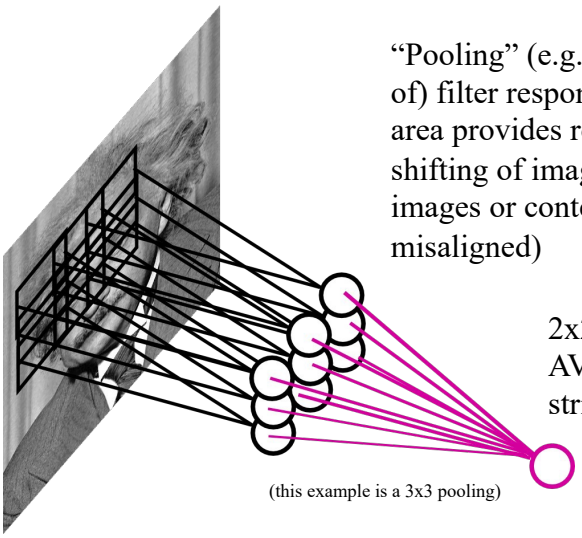
- On each feature map element (independently)
  - **ReLU**, Sigmoid, TANH
  - Also include additive bias term as parameter/weight (one per feature map in feature map volume)



97

97

Spatial Pooling



"Pooling" (e.g., taking a summary of) filter responses within local area provides robustness to slight shifting of image features (e.g., if images or content slightly misaligned)

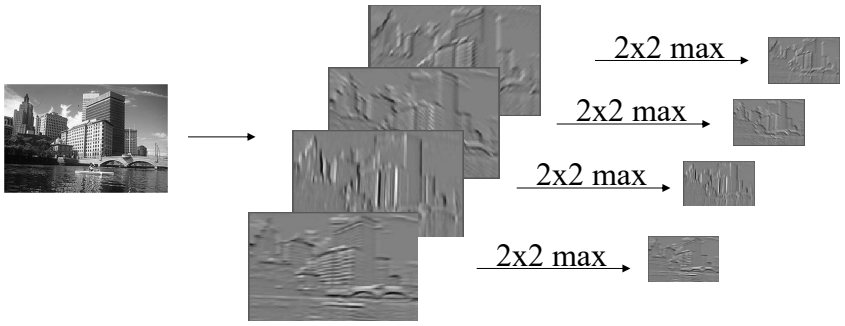
2x2 MAX (or AVERAGE) pooling w/ stride of 2 is common

(this example is a 3x3 pooling)

98

98

“Pooling Layer”

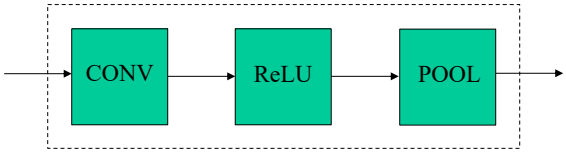


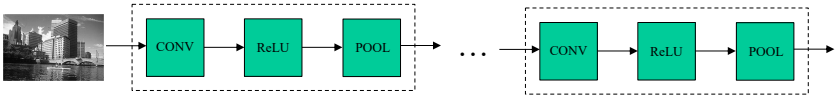
99

99

Staging

Single Stage:



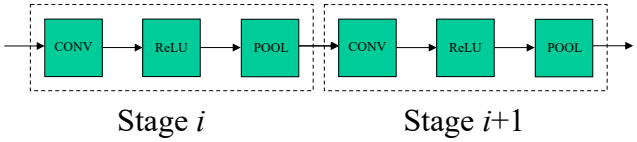


Note: Can have multiple convolution/ReLU layers before pooling

100

100

From Pooling to Next Convolution



- A convolutional filter at Stage i+1 filters across all output maps from Stage i
  - Considered as a 3-D filter (e.g., 3 x 3 x #maps)

101

101

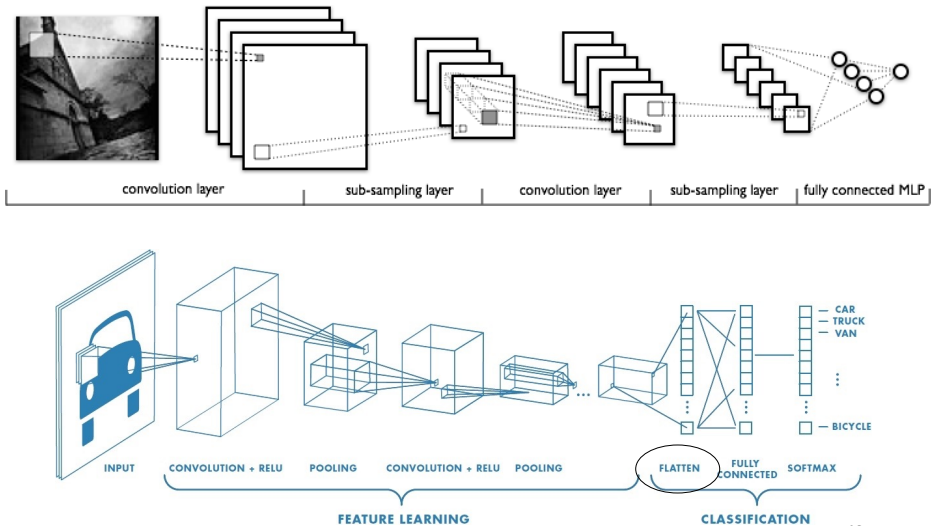
"Classification" Layers

- Lastly, for a classification network, employ fully-connected neural network at the end (1-D vector output)
- Output class labels for the input image
  - e.g., SoftMax output for hard-target class labels
  - Typically no activation function at final layer

102

102

Putting it all together…



103

"Fully-Convolutional" CNN
(FCN)

- Use convolutions (perhaps with pooling) all the way to the end
  - No fully-connected 1-D classification layer at end
- With multiple feature maps at the end, learn to output a "vector of values" per spatial location
  - Semantic Segmentation (vector of class probabilities at each pixel)

104

104

What if Input Image is Larger/Smaller?

- Method #1: Crop to fixed-size input
- Method #2: Scale to fixed-size input
  - Isotropic (if same aspect ratio, else clip out center)
  - Anisotropic (but aspect ratio will change)

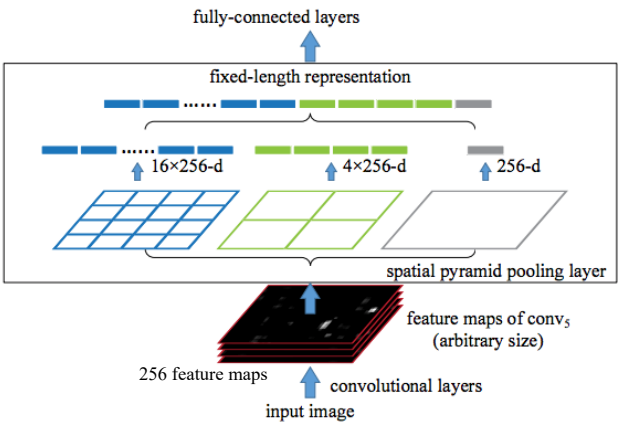
OR USE…

105

105

Spatial Pyramid Pooling

Handles varying sized images by pooling into a spatial hierarchy (e.g., 4x4, 2x2, 1x1), rasterizing (e.g., 16x1, 4x1, 1x1), then concatenating into a single vector (e.g., 21x1)



106

106

How to Train?

- Given input images and target labels
  - e.g., scene type, primary object type present
- Normal NN backpropagation techniques
- Many, many, many weights
  - Though shared weights at convolutional layer filters
  - Most weights are at final stages of a **fully-connected** classification CNN
    - For the fully-connected nodes

107

107

Input

- Mean-subtract input image (make zero mean)
  - Pixel-wise mean
    - Single aveR, aveG, aveB subtracted from each pixel
  - Could also consider "mean image" (not pixel)
- Scale each pixel (to be between around +/-1)
  - By standard deviation of pixels in dataset
  - Or by fixed range of pixels

108

108

Input Augmentation

- Augment/distort the image during training to add more variability
- Methods
  - Scale
  - Crop
  - Flip (possibly rotate)
  - Brightness (gamma)
  - Color perturbation
  - Patch dropout (set a small region to 0)
  - etc.

109

109

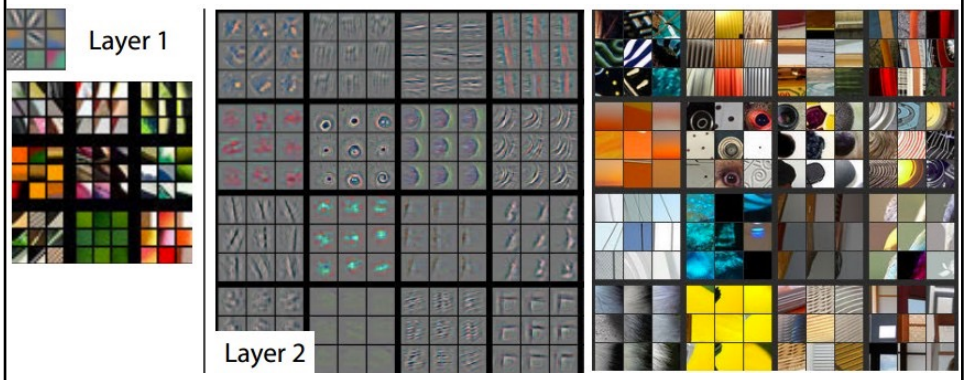
Filter Values After Training: "What are they?"

- Represent low-level, mid-level, high-level feature extraction at convolutional layers
  - The filters define the features
- Highly non-linear feature extraction, so simpler (linear) classifier at end may even be sufficient

110

110

Simple-to-Complex Filters…



111

111

"Transfer Learning" or "Network Re-use" for Different Classification Tasks

- Use trained model (on a specific task)
- Push new dataset through, and take outputs *before* final classifier
- Treat those *outputs* as input features for this new classification task
- Train a new classifier (just the classifier portion) with these features
  - Re-use of features for different tasks

112

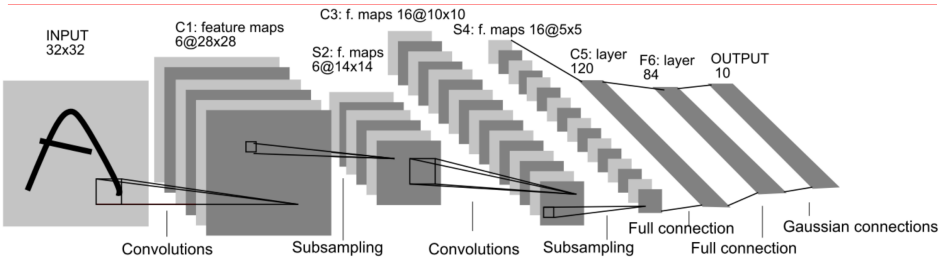
112

Classic and Modern Networks

113

Early (1998) CNN: Le-Net 5

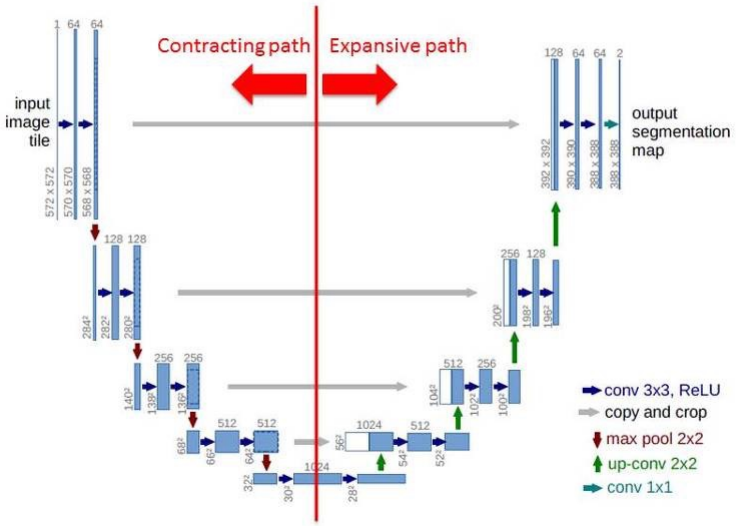
Handwritten digit recognition



114

U-Net Architecture

(…for pixel-wise classification, e.g., Semantic Segmentation)



115

115

ResNet

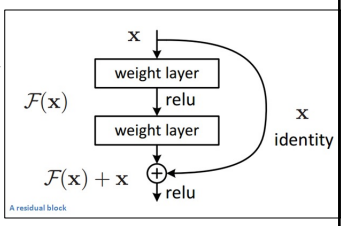
- ***Why do very deep nets perform <u>worse</u> as you keep adding layers*?**
  - Intuitively, deeper nets should perform no worse than their shallower counterparts
  - Say a net with *n* layers is learned that achieves a certain accuracy
  - A net with *n+1* layers should be able to achieve same accuracy, by copying over the same first *n* layers and performing an **identity mapping** for the last layer
    - Similarly, nets of *n+2*, *n+3*, and *n+4* layers could all continue performing **identity mappings** and achieve the same accuracy
  - In practice, however, these deeper nets almost always degrade in performance
  - Authors: *direct mappings are hard to learn*. Instead, learn a "residual" function

https://medium.com/towards-data-science/an-intuitive-guide-to-deep-network-architectures-65fdc477db41

116

116

ResNet



- An ultra-deep 152-layer CNN
- Build on "residual blocks"
  - Skip connections
- It's much easier to learn to push F(x) to 0 and leave the output as x, than to learn an **identity transformation** from scratch.
- State-of-the-art model, many/most new CNNs are still based on this approach

117

117

However…

- As gradient flows through the network, there is nothing to force it to go through residual block weights and it can avoid learning anything during training
- Therefore it is possible that there is either **only a few blocks that learn useful representations** or **many blocks share very little information with small contribution** to the final goal

118

118

Popular ResNet Sizes

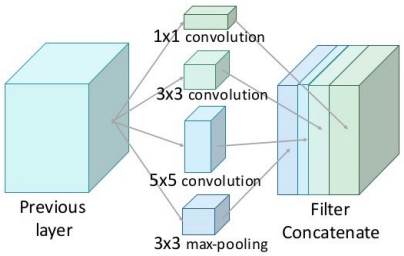
layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				

119

119

"Inception" Style Networks

- ResNet was about going deeper, the Inception network is all about going wider
- Run multiple filters over the same input map in parallel, then concatenating their results into a single output
- The next layer of the model gets to decide if (and how) to use each piece of information



120

120

ResNeXt

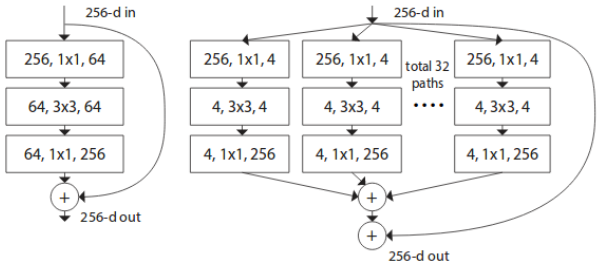


Figure 1. **Left**: A block of ResNet [14]. **Right**: A block of ResNeXt with cardinality = 32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels).

ResNet (left) and ResNeXt (right) Architecture.

121

121

EfficientNet

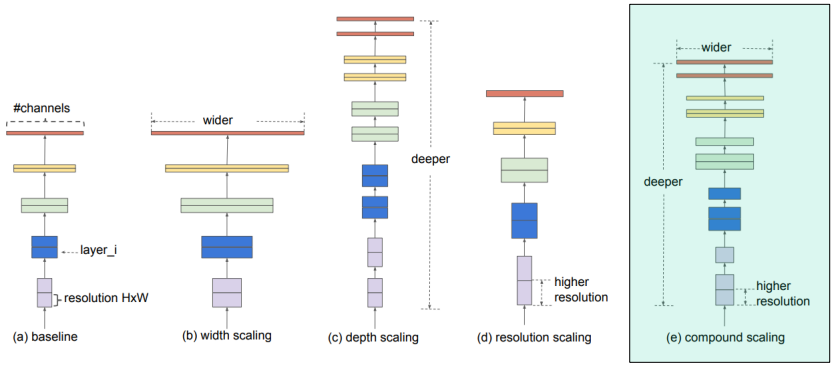


Figure 2. **Model Scaling.** (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.

Offers a method to scale up a baseline CNN to target resource constraints (larger) in a more principled way, while maintaining model efficiency, to achieve better results.

122

122

Neural Architecture Search (NAS)

- Most model architectures are designed (or borrowed) by humans, and can be ad hoc
- Neural Architecture Search methods can be used to *explore* different network architecture spaces and choose best
  - For a specific need/requirement or dataset
  - Computationally expensive

123

123

IMAGENET
Classification Challenge(s)

- Goal is to "estimate the content" of photographs for the purpose of retrieval and automatic annotation
  - ImageNet dataset has approx 10,000,000 labeled images depicting 10,000+ object categories
- 2012 classification challenge had 5 guesses
  - Algorithms produced a list of at most 5 object categories in the descending order of confidence (compared to single ground truth label for image)

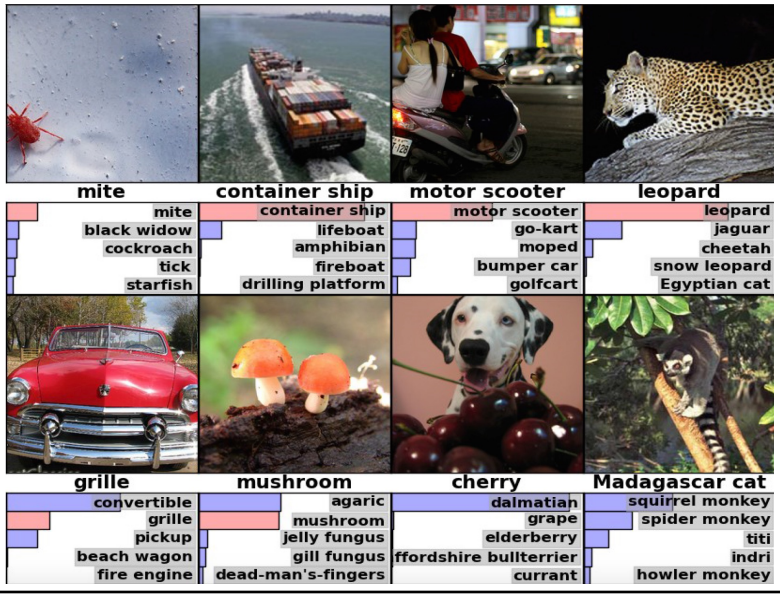
124

124

ImageNet “Hammers”

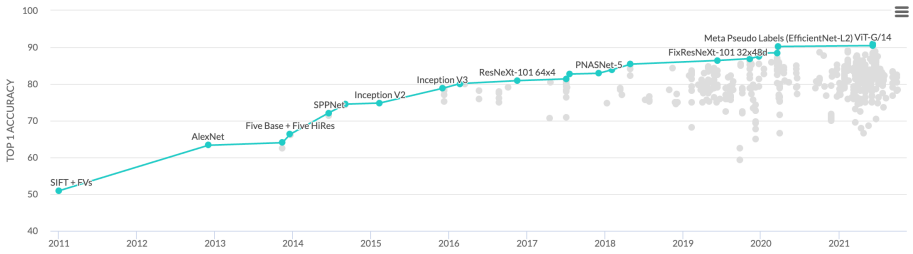
125

“5-best” Examples (AlexNet)



126

History of ImageNet Top-1 Accuracy

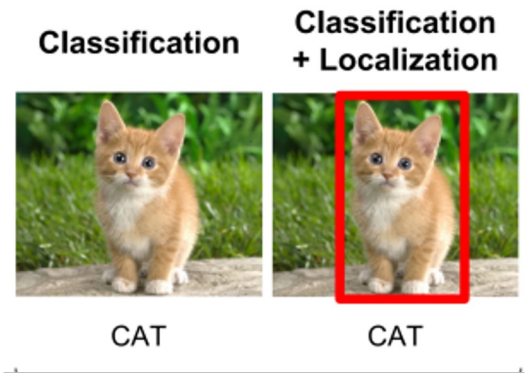


https://paperswithcode.com/sota/image-classification-on-imagenet

127

127

Object Detection
(Classification + Localization)



128

128

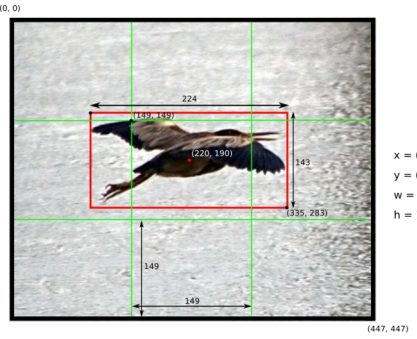
"You Only Look Once" (YOLO)

- Single end-to-end network
- Reframes the object detection problem as a single regression task
- Straight from image pixels to bounding box coordinates and class probabilities

129

129

YOLO Details



x = (220-149) / 149 = 0.48
y = (190-149) / 149 = 0.28
w = 224 / 448 = 0.50
h = 143 / 448 = 0.32

- Partitions input/output into an S x S grid of cells
- Each grid cell predicts
  - B bounding boxes (leads to predictor specialization)
  - C class probabilities
- Each bounding box predicts (x, y, w, h, confidence)
  - **(x, y)**: center of box (normalized)
  - **(w, h)**: width, height of box (normalized)
  - **confidence:** Prob(object)
- Class probabilities: Prob(class | object)
- Overall, final output is a tensor of size
  S x S x (B*5 + C)

130

130

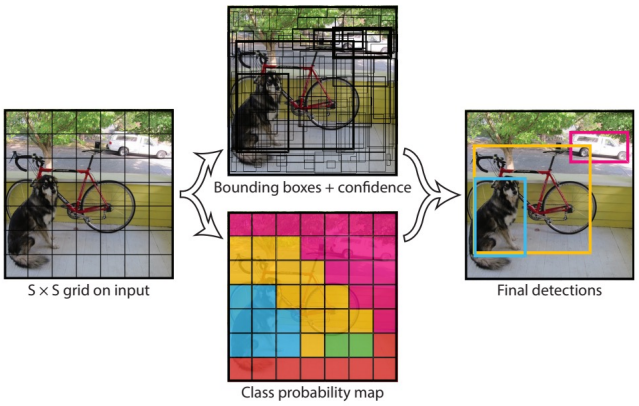


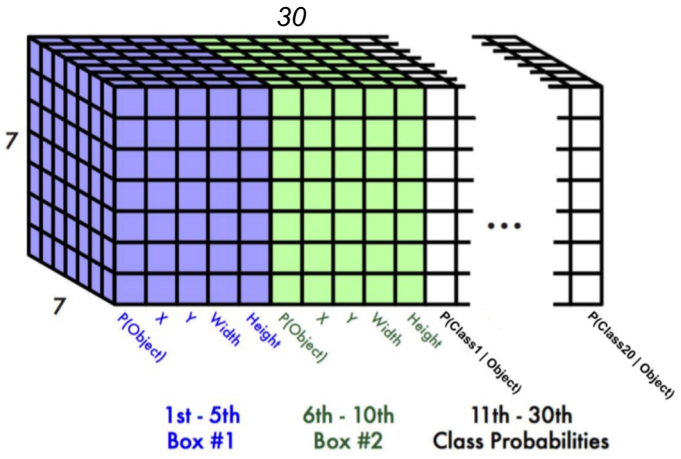
Figure 2: The Model. Our system models detection as a regression problem. It divides the image into an $S \times S$ grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an $S \times S \times (B * 5 + C)$ tensor.

131

131

Yolo Output Interpretation

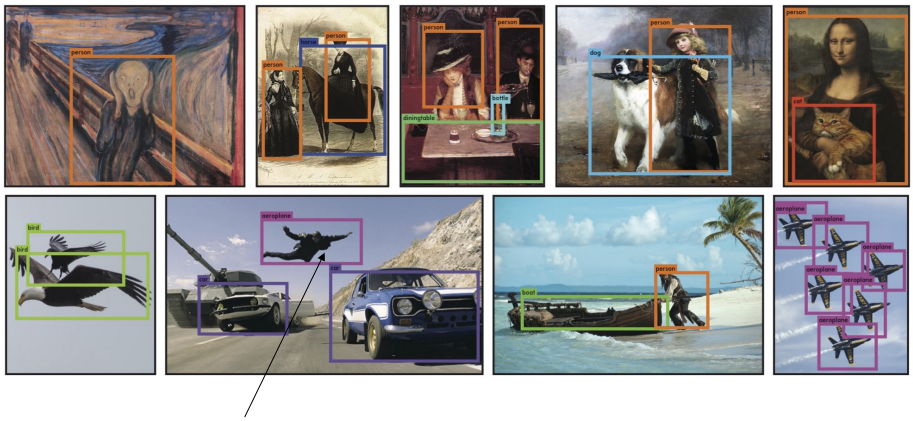
(S=7, B=2, C=20)



132

132

Example Results



133

133

Inference and Extensions

- Final prediction via post-processing
  - Uses a Non-Maximal Suppression method
  - Note: Bounding boxes that do not "confidently" describe an object (e.g., all class probabilities are below a threshold) can be ignored/removed
- YOLOv2: Refined the design and made use of predefined "anchor" boxes to improve bounding box proposal
  - Learn offsets/scales to pre-defined box sizes/aspects (priors on object shapes – learned by clustering ground truth boxes)
- YOLOv3: Further refined the model architecture and training process

Good overview:
https://jonathan-hui.medium.com/real-time-object-detection-with-yolo-yolov2-28b1b93e2088

134

134

Final Terminology to Remember…

- Deep Learning ≠ CNN
  - Other deep neural network architectures that are not CNN
- But CNN is a form of Deep Learning
  - Think of CNN ⊂ Deep Learning

135

135

Complexity of Biological Neuron

"… showed that a deep neural network requires between five and eight layers of interconnected `neurons' to represent the complexity of one single biological neuron."

https://www.quantamagazine.org/how-computationally-complex-is-a-single-neuron-20210902/

136

136

Summary

- Neural net components
- Feed-forward network
- Back-propagation for multilayer networks
- Training particulars
- Deep learning
- CNNs

But soooooo much more to talk about!

137

137