Neural Networks: Architecture

CS229: Machine Learning

winter

Stanford University, ~~Spring~~ 2024
(Adapted from slides by Matgus Telgarsky and Alexander Schwing)

Goals of this lecture

- Understand the motivation for deep neural networks
- Learn about deep neural network architecture(s) and some standard components

Goals of this lecture

- Understand the motivation for deep neural networks
- Learn about deep neural network architecture(s) and some standard components

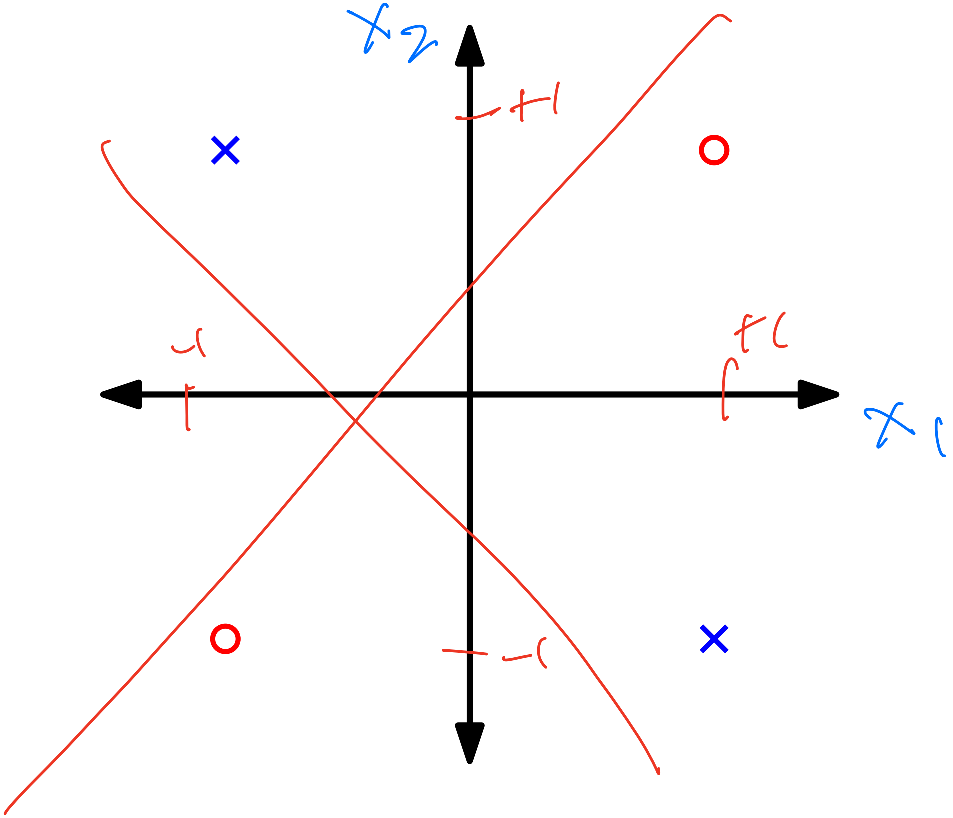
Reading material

- Course Notes, Section 7.1, 7.2
- I. Goodfellow et al.; Deep Learning; Chapters 6-9

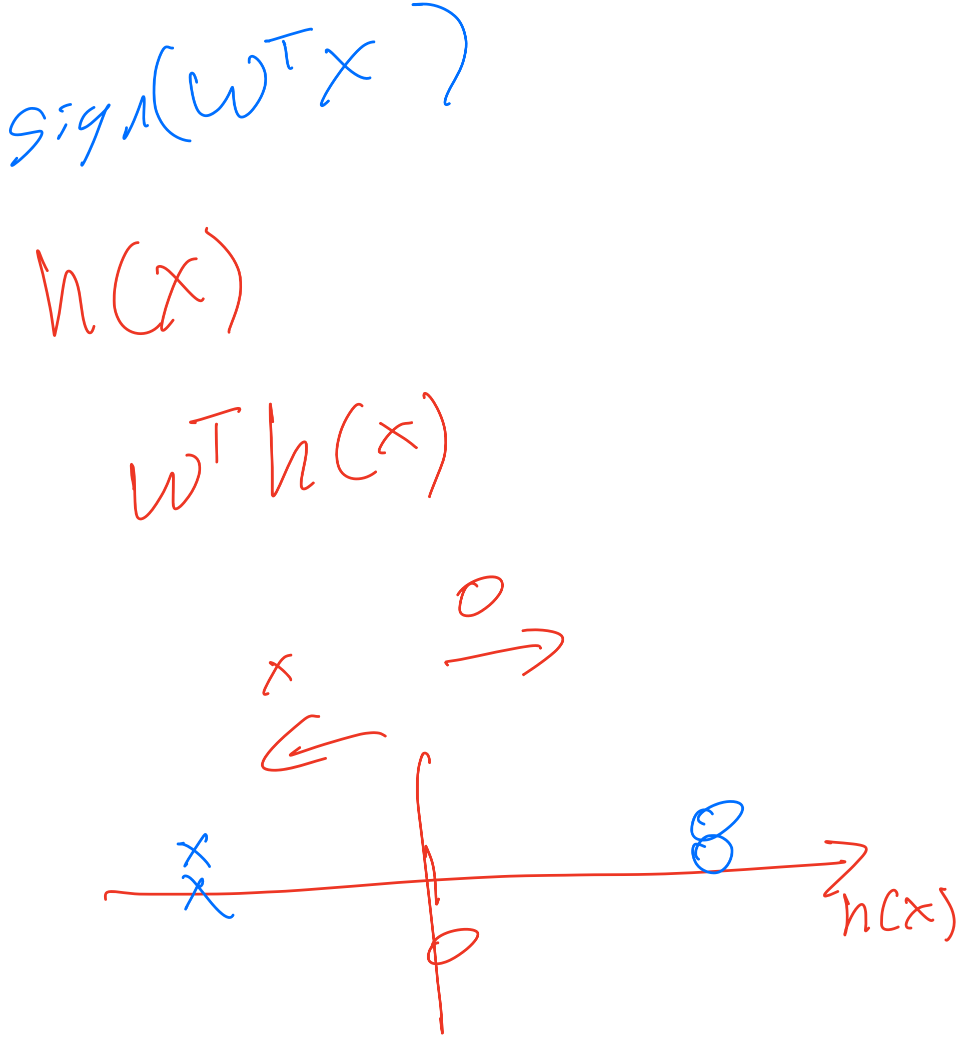
Lecture notation

Notation	Usage
$h(\cdot)$	Feature function; $\overset{\phi}{\cancel{h}}(\cdot)$ in the notes
$f(\cdot)$	Prediction function; $h(\cdot)$ in the notes
$l(\cdot, \cdot)$	Loss function; $J(\cdot)$ in the notes
$\boldsymbol{w}, \boldsymbol{W}$	Model Parameters, θ in the notes
$\boldsymbol{x}^{(i)}, \boldsymbol{x}$	Input(s)
$y^{(i)}, y$	Label(s)
λ	Regularization parameter(s); C in the notes
$\sigma(\cdot)$	Activation function, nonlinearity

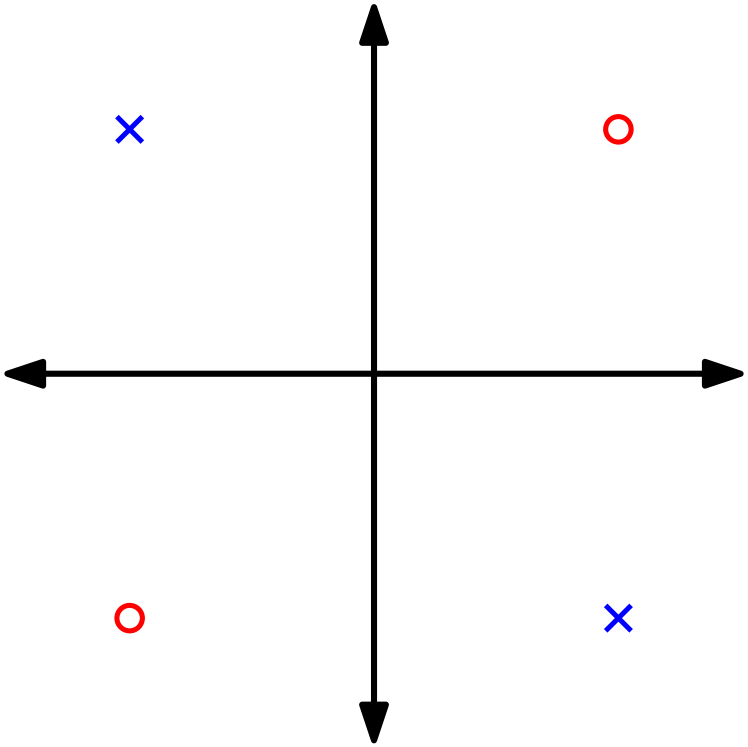
Limitations of linear predictors?



No linear separator classifies perfectly!



Limitations of linear predictors?

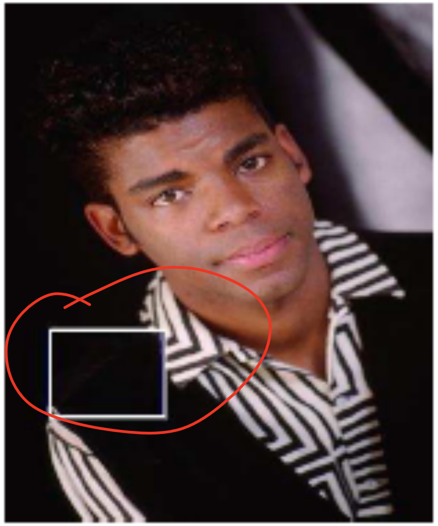


No linear separator classifies perfectly!

Feature Transformation:
use features $h(\boldsymbol{x}) := \boldsymbol{x}_1^\top \boldsymbol{x}_2$,
with prediction $y = \text{sgn}\left(\boldsymbol{w}^\top h(\boldsymbol{x})\right)$.

Real-world Application: **Edge/Boundary detection:**

What are y, x?

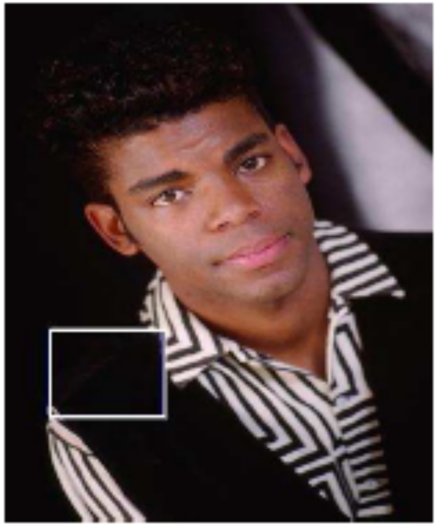


$y :=$ background -1
foreground $+1$

$x :=$ pixels
(intensity, color)

Real-world Application: Edge/Boundary detection:

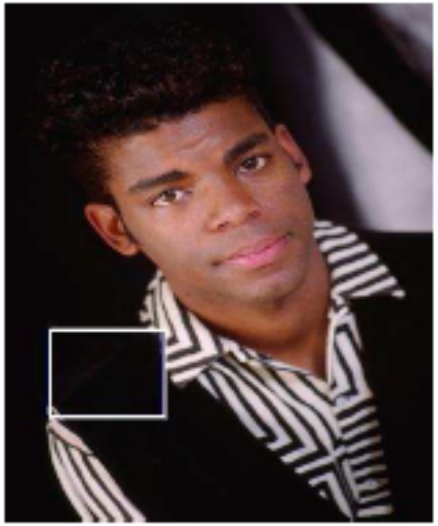
Issues?



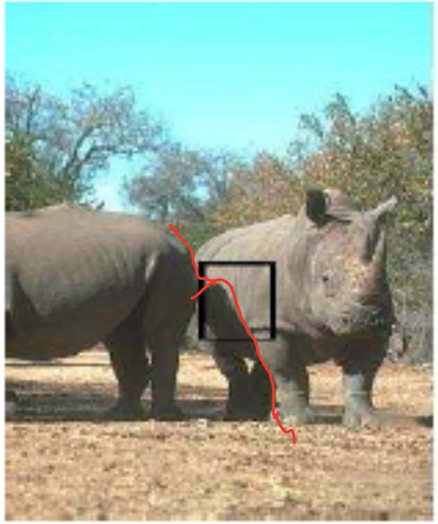
Poor contrast

Real-world Application: Edge/Boundary detection:

Issues?

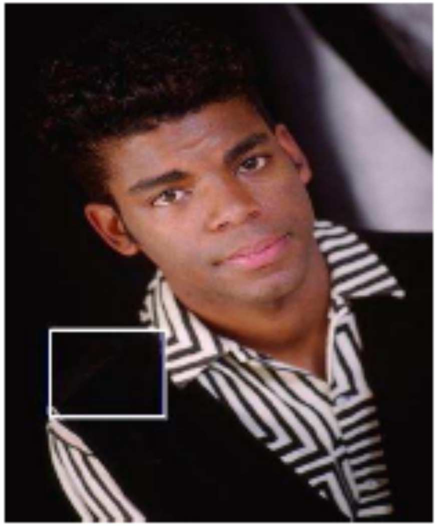


Poor contrast

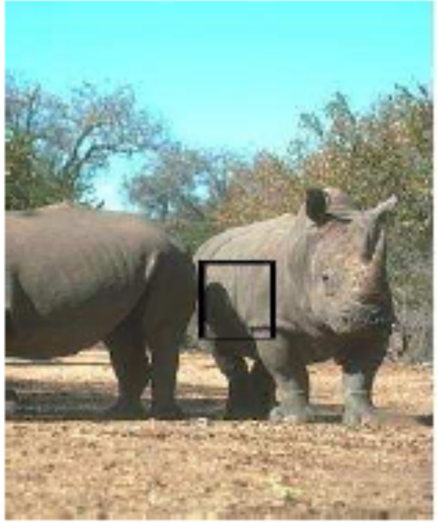


Real-world Application: Edge/Boundary detection:

Issues?



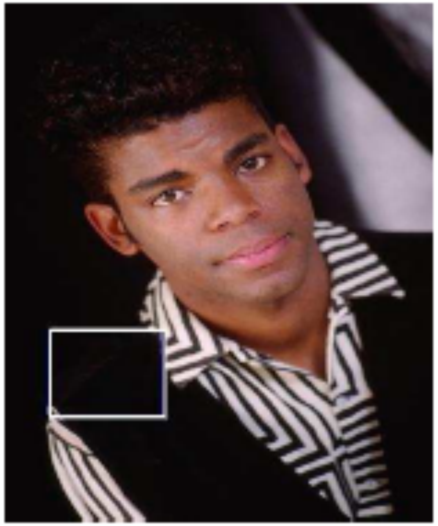
Poor contrast



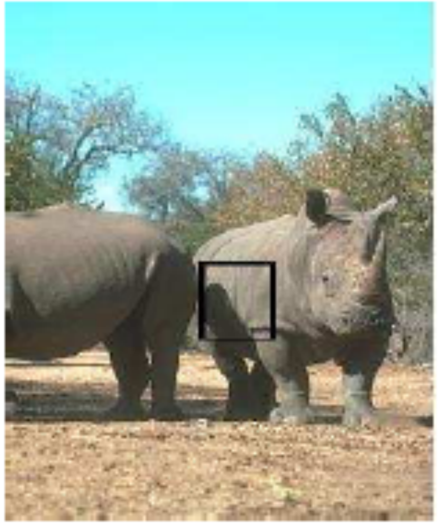
Shadow

Real-world Application: Edge/Boundary detection:

Issues?



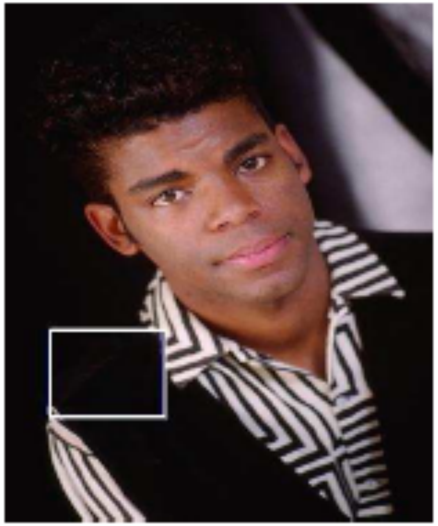
Poor contrast



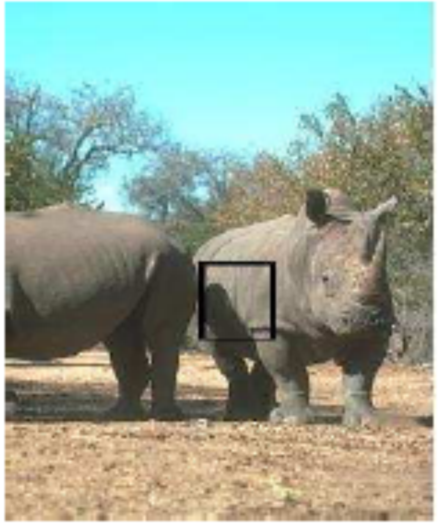
Shadow

Real-world Application: Edge/Boundary detection:

Issues?



Poor contrast

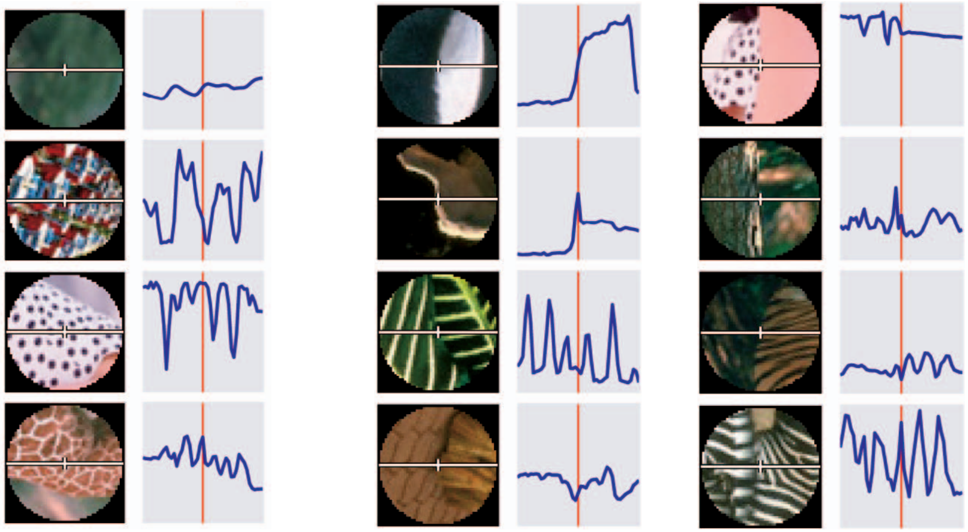


Shadow

Texture

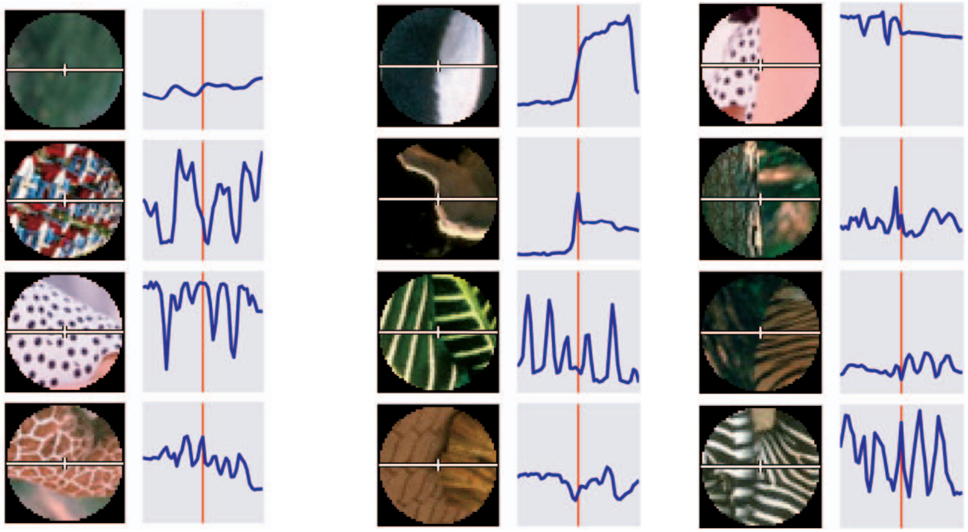
Edge/Boundary detection: Why is it so difficult?

Let’s look at a local image region and the corresponding intensities:



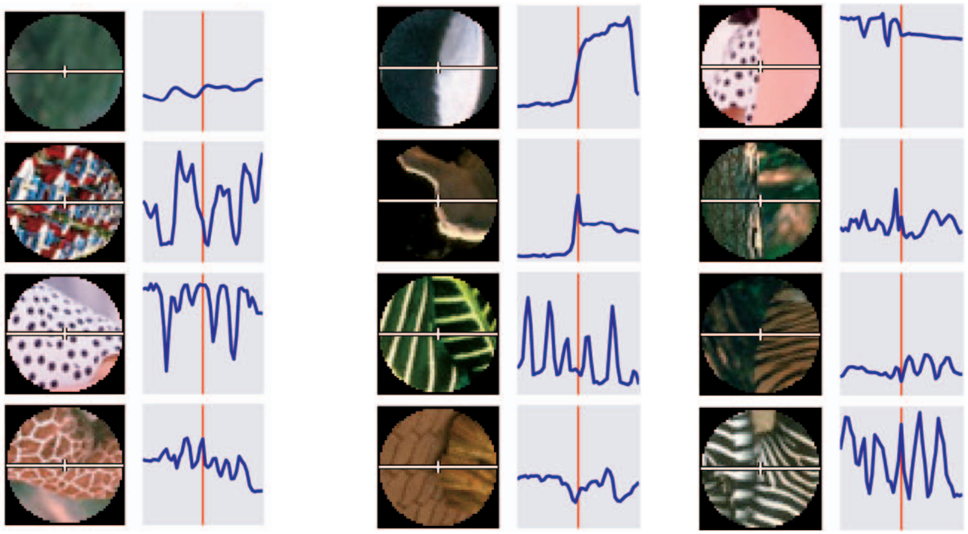
Edge/Boundary detection: Why is it so difficult?

Let's look at a local image region and the corresponding intensities:



Non-Boundaries

Edge/Boundary detection: Why is it so difficult?
Let's look at a local image region and the corresponding intensities:



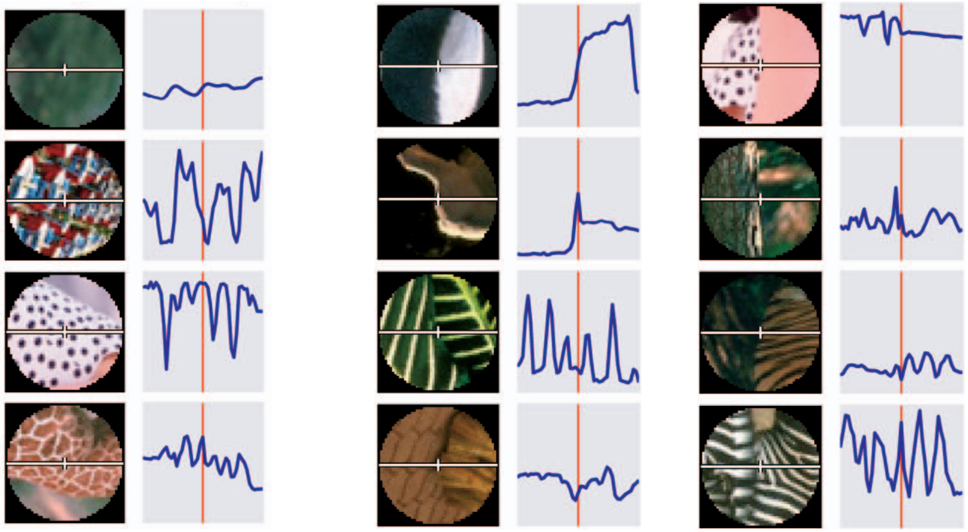
Non-Boundaries Boundaries

$h(x)$ = pixel intensity

⟶ avg local intensity

Edge/Boundary detection: Why is it so difficult?

Let's look at a local image region and the corresponding intensities:



Non-Boundaries Boundaries

Intensity cue is not necessarily a good indicator for boundaries.

Martin et al., *Learning to detect natural Image Boundaries ...*, PAMI 2004

Edge/Boundary detection: What other image cues could be helpful?

Martin et al., *Learning to detect natural Image Boundaries ...*, PAMI 2004

Edge/Boundary detection: What other image cues could be helpful?

- Brightness gradient (BG)

Martin et al., *Learning to detect natural Image Boundaries ...*, PAMI 2004

Edge/Boundary detection: What other image cues could be helpful?

- Brightness gradient (BG)
- Color gradient (CG)

Martin et al., *Learning to detect natural Image Boundaries ...*, PAMI 2004

Edge/Boundary detection: What other image cues could be helpful?

- Brightness gradient (BG)
- Color gradient (CG)
- Texture gradient (TG)

Martin et al., *Learning to detect natural Image Boundaries ...*, PAMI 2004

Edge/Boundary detection: What other image cues could be helpful?

- Brightness gradient (BG)
- Color gradient (CG)
- Texture gradient (TG)
- ...

Martin et al., *Learning to detect natural Image Boundaries ...*, PAMI 2004

Martin et al., *Learning to detect natural Image Boundaries ...*, PAMI 2004

How to combine all those cues?

$h(x_i) = \begin{bmatrix} BG \\ CG \\ \vdots \end{bmatrix}$ $\quad w^T h(x) = f(x)$

Martin et al., *Learning to detect natural Image Boundaries ...*, PAMI 2004

How to combine all those cues? Learn a linear combination of cues

- What is $y^{(i)}$?

- What is $y^{(i)}$? Annotated pixel label

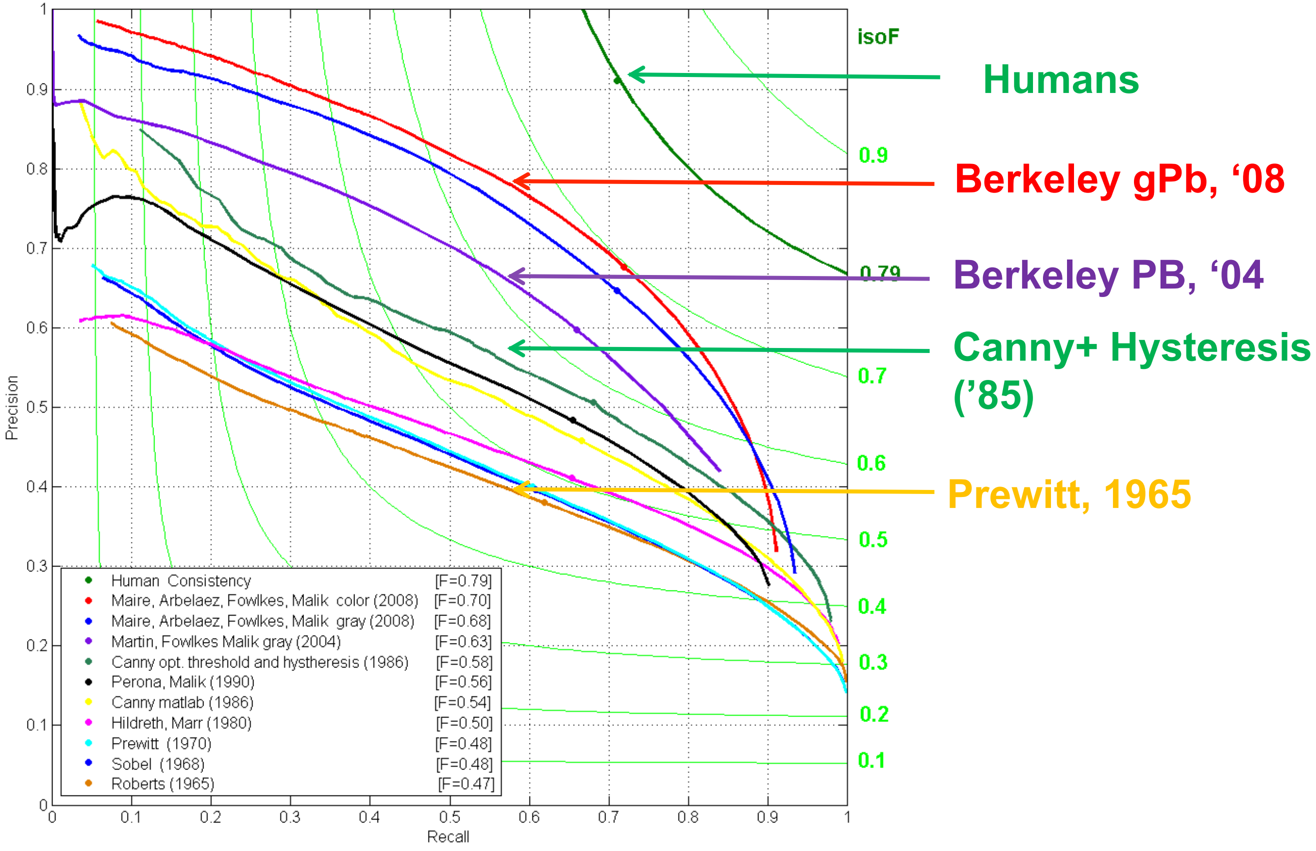
- What is $y^{(i)}$? Annotated pixel label
- What is $x^{(i)}$?

- What is $y^{(i)}$? Annotated pixel label
- What is $x^{(i)}$? Image pixel

- What is $y^{(i)}$? Annotated pixel label
- What is $x^{(i)}$? Image pixel
- What is $h(x^{(i)})$?

- What is $y^{(i)}$? Annotated pixel label
- What is $x^{(i)}$? Image pixel
- What is $h(x^{(i)})$? Vector of features computed in the **neighborhood** of pixel i, e.g., intensity, texture gradient, oriented gradient etc.

Boundary detection performance:



Key Takeaways:

- Selecting good features is often key to the performance of an ML algorithm.

Key Takeaways:

- Selecting good features is often key to the performance of an ML algorithm.
- Most common approach is to apply known heuristics that work well from (community/expert) experience.

Key Takeaways:

- Selecting good features is often key to the performance of an ML algorithm.
- Most common approach is to apply known heuristics that work well from (community/expert) experience.
- Caution: when/how can adding features hurt performance?

Key Takeaways:

- Selecting good features is often key to the performance of an ML algorithm.
- Most common approach is to apply known heuristics that work well from (community/expert) experience.
- Caution: when/how can adding features hurt performance?
- Next, we will discuss how deep learning can help automate feature extraction.

Neural networks via *features*.

To make a linear predictor nonlinear in x, we rely upon feature mapping h:

$$\boldsymbol{w}^\top \boldsymbol{x} \qquad \text{becomes} \qquad \boldsymbol{w}^\top h(\boldsymbol{x}).$$

We are at the mercy of the quality of h.

Neural networks via *features*.

To make a linear predictor nonlinear in $\boldsymbol{x}$, we rely upon feature mapping h:

$$\boldsymbol{w}^\top \boldsymbol{x} \qquad \text{becomes} \qquad \boldsymbol{w}^\top h(\boldsymbol{x}).$$

We are at the mercy of the quality of h.

Why not *learn* h?

Neural networks via *features*.

To make a linear predictor nonlinear in $\boldsymbol{x}$, we rely upon feature mapping h:

$$\boldsymbol{w}^\top \boldsymbol{x} \qquad \text{becomes} \qquad \boldsymbol{w}^\top h(\boldsymbol{x}).$$

We are at the mercy of the quality of h.

$\ell(w) = -\log P(y \mid x; w)$

Why not *learn* h? e.g.,

$$\min_{\boldsymbol{w}} \frac{1}{n} \sum_{i=1}^{n} \ell\left(y^{(i)}, \boldsymbol{w}^\top \boldsymbol{x}^{(i)}\right) \qquad \text{becomes} \qquad \min_{\boldsymbol{w}, h} \frac{1}{n} \sum_{i=1}^{n} \ell\left(y^{(i)}, \boldsymbol{w}^\top h(\boldsymbol{x}^{(i)})\right)$$

Neural networks as iterated linear prediction (part 1).

Natural choice: build feature maps out of linear predictors!

Neural networks as iterated linear prediction (part 1).

Natural choice: build feature maps out of linear predictors!

$$\boldsymbol{w}^\top \boldsymbol{x} \qquad \text{becomes} \qquad \boldsymbol{v}^\top h(\boldsymbol{x}) \quad \text{where } h(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}$$

with $\boldsymbol{w} \in \mathbb{R}^d, \boldsymbol{v} \in \mathbb{R}^m, \boldsymbol{A} \in \mathbb{R}^{m \times d}, \boldsymbol{b} \in \mathbb{R}^m$.

Neural networks as iterated linear prediction (part 1).

Natural choice: build feature maps out of linear predictors!

$$\boldsymbol{w}^\top \boldsymbol{x} \qquad \text{becomes} \qquad \boldsymbol{v}^\top h(\boldsymbol{x}) \quad \text{where } h(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}$$

with $\boldsymbol{w} \in \mathbb{R}^d, \boldsymbol{v} \in \mathbb{R}^m, \boldsymbol{A} \in \mathbb{R}^{m \times d}, \boldsymbol{b} \in \mathbb{R}^m.$

There is something wrong with this!

Neural networks as iterated linear prediction (part 1).

Natural choice: build feature maps out of linear predictors!

$$\boldsymbol{w}^\top \boldsymbol{x} \qquad \text{becomes} \qquad \boldsymbol{v}^\top h(\boldsymbol{x}) \qquad \text{where } h(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}$$

with $\boldsymbol{w} \in \mathbb{R}^d, \boldsymbol{v} \in \mathbb{R}^m, \boldsymbol{A} \in \mathbb{R}^{m \times d}, \boldsymbol{b} \in \mathbb{R}^m$.

There is something wrong with this!

Gained no additional representation flexibility!

$$\boldsymbol{v}^\top(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}) = (\boldsymbol{A}^\top \boldsymbol{v})^\top \boldsymbol{x} + \boldsymbol{v}^\top \boldsymbol{b}.$$

w

Neural networks as iterated linear prediction (part 1).

Natural choice: build feature maps out of linear predictors!

$$\boldsymbol{w}^\top \boldsymbol{x} \qquad \text{becomes} \qquad \boldsymbol{v}^\top h(\boldsymbol{x}) \quad \text{where } h(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}$$

with $\boldsymbol{w} \in \mathbb{R}^d, \boldsymbol{v} \in \mathbb{R}^m, \boldsymbol{A} \in \mathbb{R}^{m \times d}, \boldsymbol{b} \in \mathbb{R}^m$.

There is something wrong with this!

Gained no additional representation flexibility!

$$\boldsymbol{v}^\top(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}) = (\boldsymbol{A}^\top \boldsymbol{v})^\top \boldsymbol{x} + \boldsymbol{v}^\top \boldsymbol{b}.$$

$$\sigma(v) = \begin{bmatrix} \sigma(v_1) \\ \sigma(v_2) \\ \vdots \\ \sigma(v_m) \end{bmatrix}$$

Fix: introduce **nonlinearity/transfer/activation** $\sigma : \mathbb{R}^m \to \mathbb{R}^m$:

$$h(\boldsymbol{x}) := \sigma\left(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}\right).$$

Neural networks as iterated linear prediction (part 2).

We will predict as

$h(x; A, b, \sigma)$ ← fit

$$\boldsymbol{w}^\top h(\boldsymbol{x}) \qquad \text{where } h(\boldsymbol{x}) = \sigma\left(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}\right).$$

We will train as

$$\min_{\boldsymbol{w}\in\mathbb{R}^m, \boldsymbol{A}\in\mathbb{R}^{m\times d}, \boldsymbol{b}\in\mathbb{R}^m} \frac{1}{n}\sum_{i=1}^{n} \ell\left(y^{(i)}, \boldsymbol{w}^\top \sigma\left(\boldsymbol{A}\boldsymbol{x}^{(i)} + \boldsymbol{b}\right)\right).$$

Question: which training procedure? Gradient Desent

OR SGD

Neural networks as iterated linear prediction (part 2).

We will predict as

$$\boldsymbol{w}^\top h(\boldsymbol{x}) \qquad \text{where } h(\boldsymbol{x}) = \sigma\left(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}\right).$$

We will train as

$$\min_{\boldsymbol{w}\in\mathbb{R}^m, \boldsymbol{A}\in\mathbb{R}^{m\times d}, \boldsymbol{b}\in\mathbb{R}^m} \frac{1}{n}\sum_{i=1}^{n} \ell\left(y^{(i)}, \boldsymbol{w}^\top \sigma\left(\boldsymbol{A}\boldsymbol{x}^{(i)} + \boldsymbol{b}\right)\right).$$

Question: which training procedure?

Why stop there? We can also do

$$\boldsymbol{w}^\top \sigma_1\left(\boldsymbol{A}^{[1]} h(\boldsymbol{x}) + \boldsymbol{b}^{[1]}\right) \qquad \text{where } h(\boldsymbol{x}) = \sigma_2\left(\boldsymbol{A}^{[2]}\boldsymbol{x} + \boldsymbol{b}^{[2]}\right),$$

and iterate further.

Neural networks as iterated linear prediction (part 2).

We will predict as

$$\boldsymbol{w}^\top h(\boldsymbol{x}) \qquad \text{where } h(\boldsymbol{x}) = \sigma\left(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}\right).$$

We will train as

$$\min_{\boldsymbol{w}\in\mathbb{R}^m, \boldsymbol{A}\in\mathbb{R}^{m\times d}, \boldsymbol{b}\in\mathbb{R}^m} \frac{1}{n}\sum_{i=1}^{n} \ell\left(y^{(i)}, \boldsymbol{w}^\top \sigma\left(\boldsymbol{A}\boldsymbol{x}^{(i)} + \boldsymbol{b}\right)\right).$$

Question: which training procedure?

Why stop there? We can also do

$$\boldsymbol{w}^\top \sigma_1\left(\boldsymbol{A}^{[1]} h(\boldsymbol{x}) + \boldsymbol{b}^{[1]}\right) \qquad \text{where } h(\boldsymbol{x}) = \sigma_2\left(\boldsymbol{A}^{[2]}\boldsymbol{x} + \boldsymbol{b}^{[2]}\right),$$

and iterate further. *This is a deep neural network.*

Neural networks as functions.

A linear predictor (**one layer network**) has the form

$$\boldsymbol{w}^\top \boldsymbol{x}.$$

A **two layer network** has the form

$$\boldsymbol{x} \mapsto \boldsymbol{w}^\top \sigma_1\left(\boldsymbol{A}^{[1]}\boldsymbol{x} + \boldsymbol{b}^{[1]}\right).$$

$h(x)$

$\to A^{[i]}, b^{[i]}$

Iterating, a **multi-layer network** has the form

$$\boldsymbol{w}^\top \sigma_1\left(\boldsymbol{A}^{[1]}\sigma_2\left(\cdots \boldsymbol{A}^{[L-2]}\sigma_{L-1}\left(\boldsymbol{A}^{[L-1]}\boldsymbol{x} + \boldsymbol{b}^{[L-1]}\right) + \boldsymbol{b}^{[L-2]}\cdots\right) + \boldsymbol{b}^{[1]}\right).$$

$h(x)$

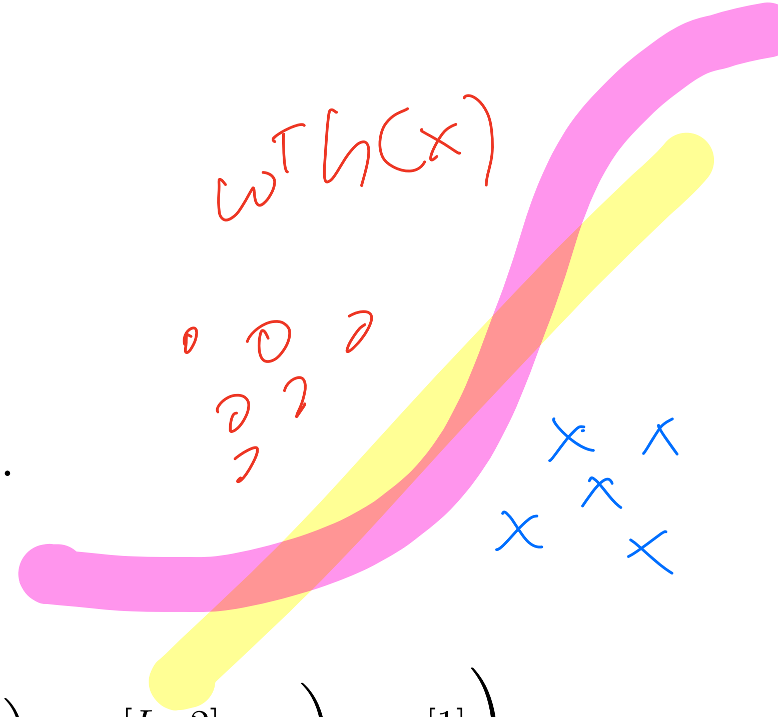
$\to$ Parameters $\{A^{[i]}, b^{[i]}\}$

Neural networks as functions.

A linear predictor (**one layer network**) has the form

$$\boldsymbol{w}^\top \boldsymbol{x}.$$

$w^T h(x)$



A **two layer network** has the form

$$\boldsymbol{x} \mapsto \boldsymbol{w}^\top \sigma_1 \left(\boldsymbol{A}^{[1]} \boldsymbol{x} + \boldsymbol{b}^{[1]} \right).$$

Iterating, a **multi-layer network** has the form

$$\boldsymbol{w}^\top \sigma_1 \left(\boldsymbol{A}^{[1]} \sigma_2 \left(\cdots \boldsymbol{A}^{[L-2]} \sigma_{L-1} \left(\boldsymbol{A}^{[L-1]} \boldsymbol{x} + \boldsymbol{b}^{[L-1]} \right) + \boldsymbol{b}^{[L-2]} \cdots \right) + \boldsymbol{b}^{[1]} \right).$$

Optimization now takes the form

$$\min_{\boldsymbol{w}, \boldsymbol{A}^{[1]}, \ldots, \boldsymbol{A}^{[L-1]}, \boldsymbol{b}_1, \ldots, \boldsymbol{b}_{L-1}} \frac{1}{n} \sum_{i=1}^{n} \ell \left(y^{(i)}, \boldsymbol{w}^\top \sigma_1 \left(\cdots \sigma_{L-1} \left(\boldsymbol{A}^{[L-1]} \boldsymbol{x}^{(i)} + \boldsymbol{b}^{[L-1]} \right) \cdots \right) \right).$$

Classical formulation of neural networks as graphs.

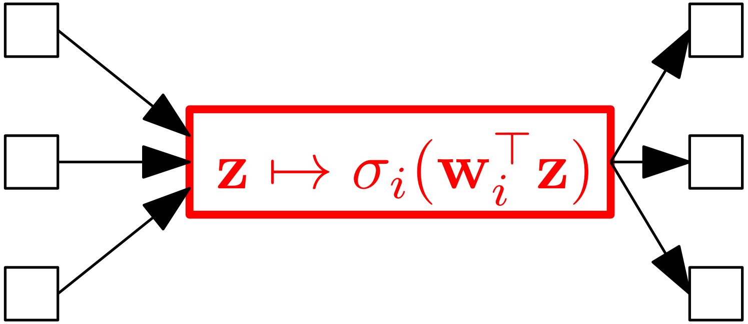
(Computation graphs in tensorflow and other software differ slightly.)

Classical formulation of neural networks as graphs.

(Computation graphs in tensorflow and other software differ slightly.)

Node j in this graph:

- Collects a vector $\boldsymbol{z}$ from its in-edges;
- Computes $\sigma_j(\boldsymbol{w}^{[j]\top}\boldsymbol{z} + b^{[j]})$;
- Propagates this value along its out-edges.



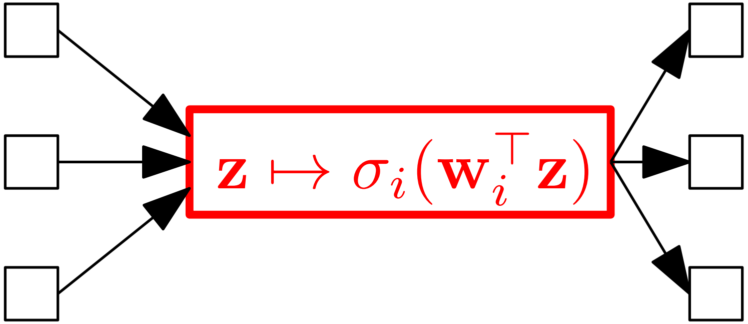
one layer

Classical formulation of neural networks as graphs.

(Computation graphs in tensorflow and other software differ slightly.)

Node j in this graph:

- Collects a vector $\boldsymbol{z}$ from its in-edges;
- Computes $\sigma_j(\boldsymbol{w}^{[j]\top}\boldsymbol{z} + b^{[j]})$;
- Propagates this value along its out-edges.



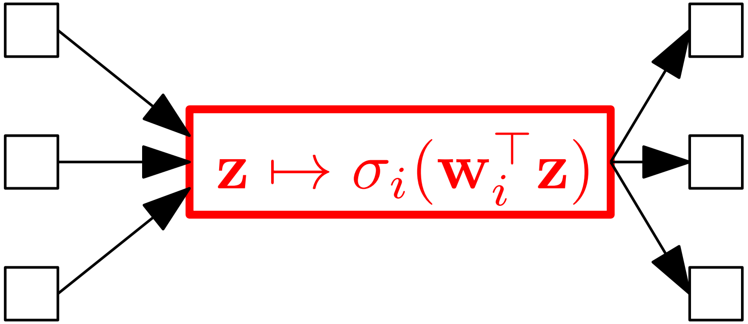
Computation of whole network can be written this way.

Classical formulation of neural networks as graphs.

(Computation graphs in tensorflow and other software differ slightly.)

Node j in this graph:

- Collects a vector z from its in-edges;
- Computes $\sigma_j(\boldsymbol{w}^{[j]\top}\boldsymbol{z} + b^{[j]})$;
- Propagates this value along its out-edges.



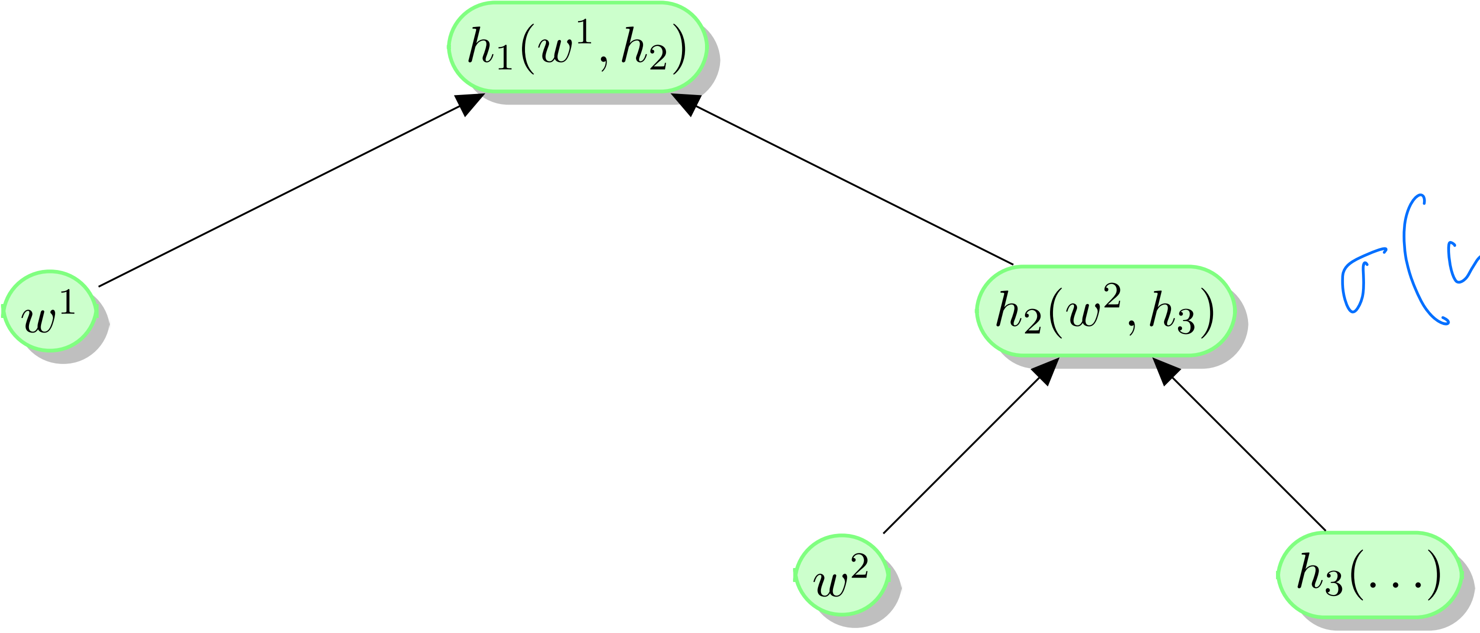
Computation of whole network can be written this way.

Tensorflow computation graphs: everything needed to train is in the graph; e.g., parameters are also assigned to nodes.

Internal representation used by deep net packages

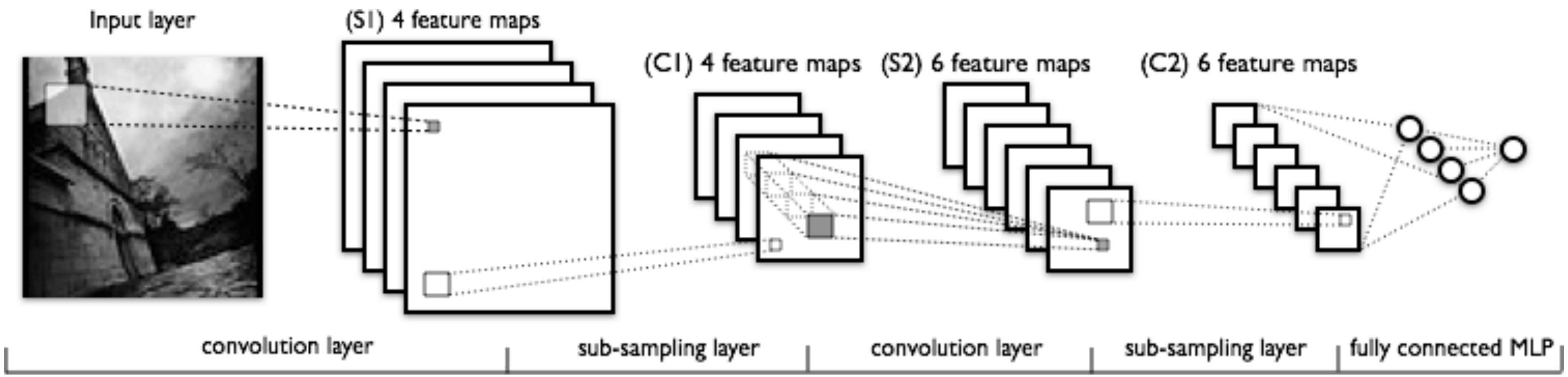
$$f(\boldsymbol{w}, x, y) = h_1(w^1, h_2(w^2, h_3(\ldots)))$$

Nodes are weights, data, and functions



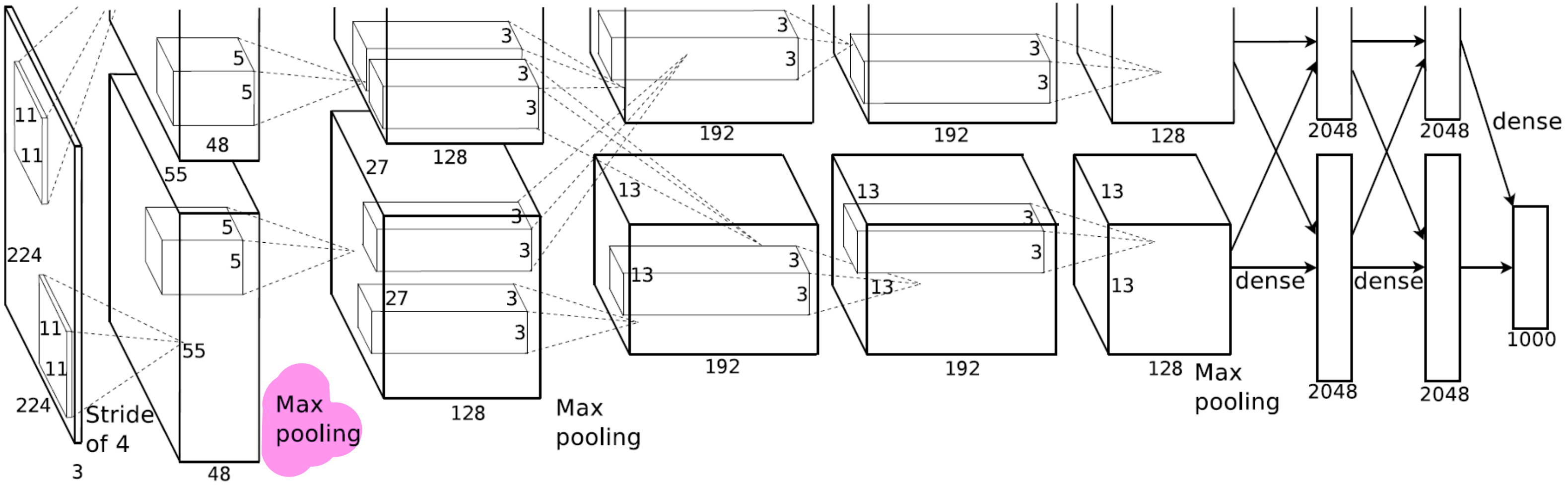
$\sigma(w^{2T} h_3())$

Example function architecture: LeNet



Decreasing spatial resolution and the increasing number of channels (dimension of feature maps)

Example function architecture: AlexNet



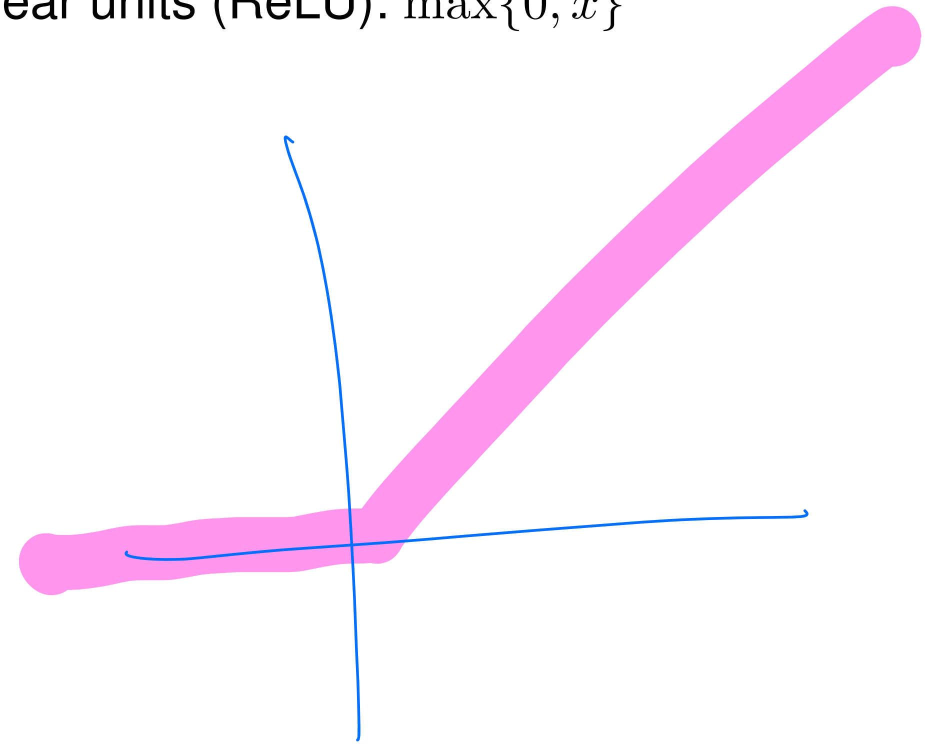
Decreasing spatial resolution and the increasing number of channels (dimension of feature maps)

Why is the output 1000-dimensional?

Common functions/layers in a deep network

Common functions/layers in a deep network

- Univariate activations e.g. rectified linear units (ReLU): $\max\{0, x\}$
- Fully connected layers
- Convolution layers
- Maximum-/Average- pooling
- Soft-max layer
- Dropout



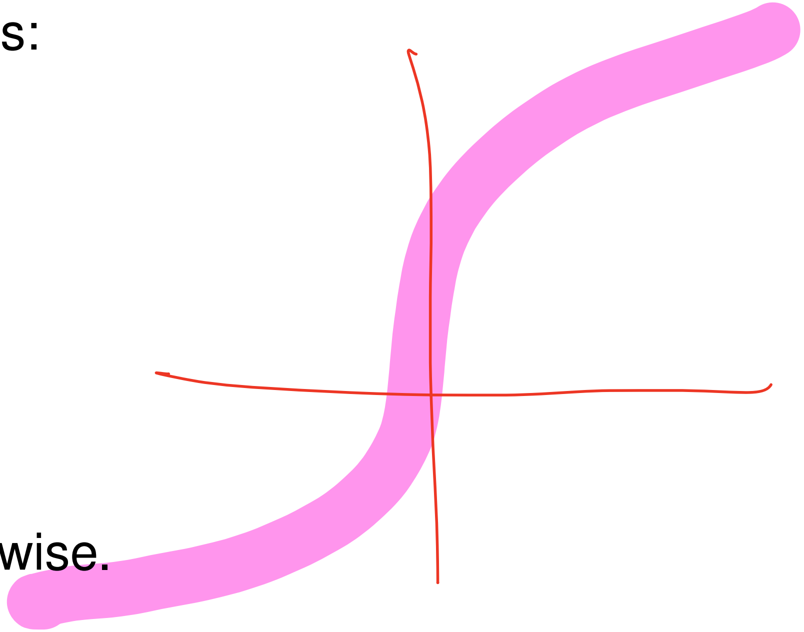
Neural network (univariate) activations.

We mentioned that **nodes** compute

$$\sigma\left(\boldsymbol{v}^\top \boldsymbol{z}\right),$$

where *activation/transfer/nonlinearity* $\sigma : \mathbb{R} \to \mathbb{R}$ is:

- ReLU (Rectified Linear Unit) $\max\{0, z\}$;
- Sigmoid $\frac{1}{1+\exp(-z)}$;
- $\dots$.



For

$$\sigma(\boldsymbol{A}\boldsymbol{z} + \boldsymbol{b}),$$

this implies applying the univariate σ coordinate-wise.

Fully connected layer

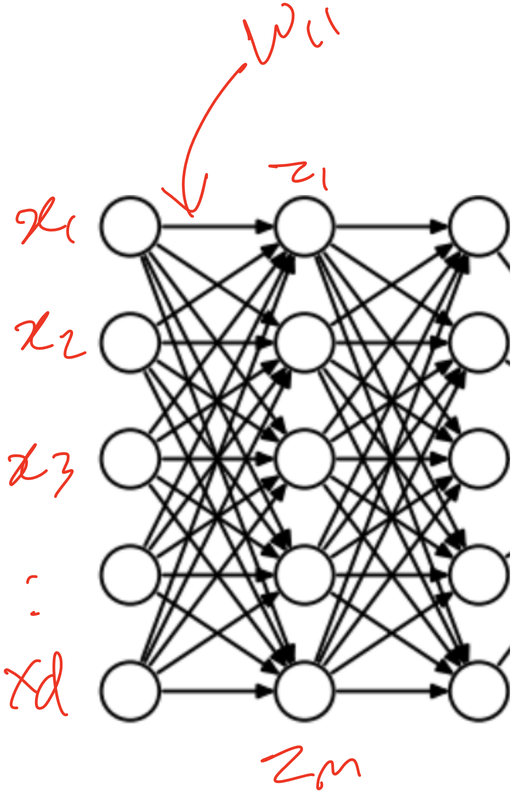
$\mathbb{R}^{m \times d}$

$\in \mathbb{R}^d$

$$\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$

Trainable parameters $\boldsymbol{w}$:

- Weight matrix
- Bias

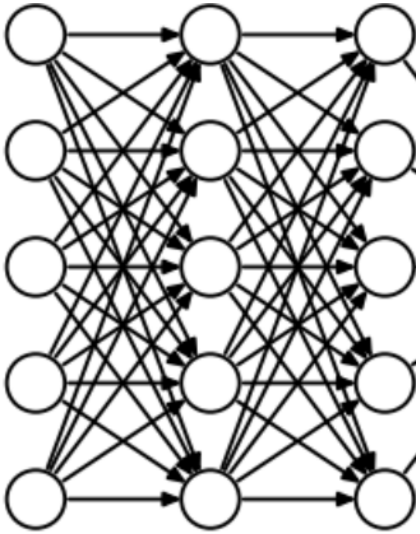


Fully connected layer

$$\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$

Trainable parameters $\boldsymbol{w}$:

- Weight matrix
- Bias



What's an issue with fully connected layers?

Issue with fully connected layers

- Suppose the input is an image of size 256×256
- Let the output of this layer have identical size
- How many weights are necessary?

$(256 \times 256)^2$

Issue with fully connected layers

- Suppose the input is an image of size 256×256
- Let the output of this layer have identical size
- How many weights are necessary?

$$(256 \times 256)^2 = 2^{32} = 4,294,967,296$$

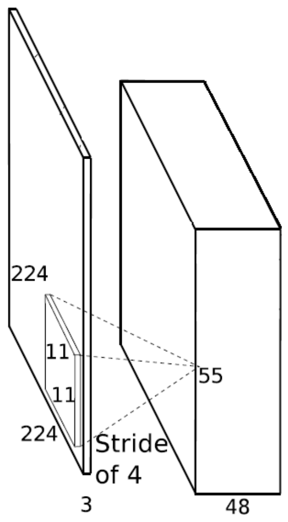
Issue with fully connected layers

- Suppose the input is an image of size 256×256
- Let the output of this layer have identical size
- How many weights are necessary?

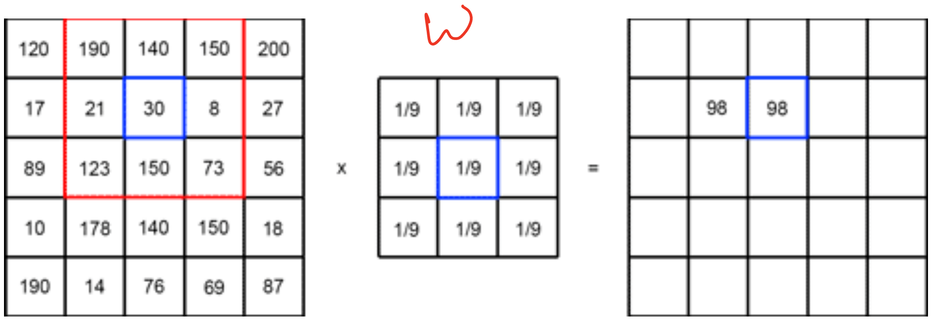
$$(256 \times 256)^2 = 2^{32} = 4,294,967,296$$

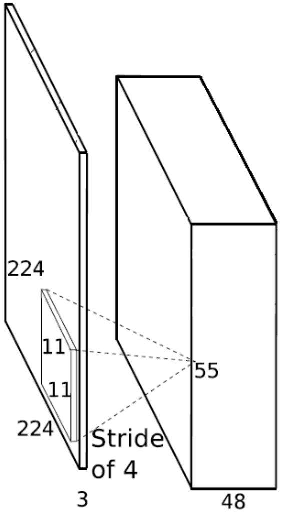
- A solution: share weights

Convolutions

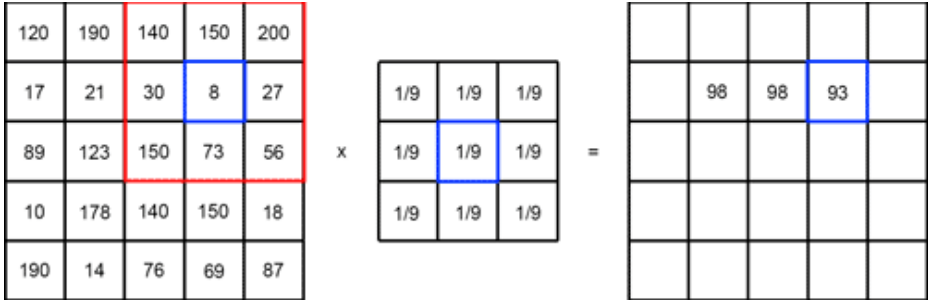


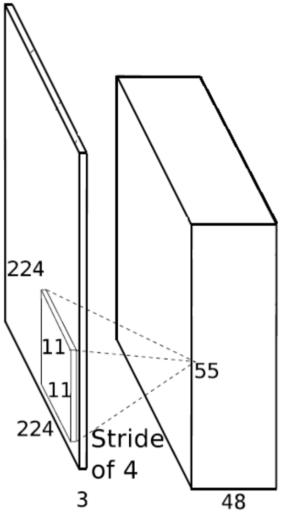
Convolutions



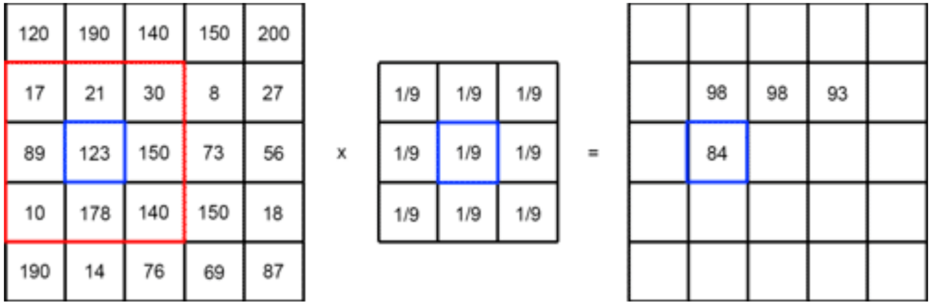


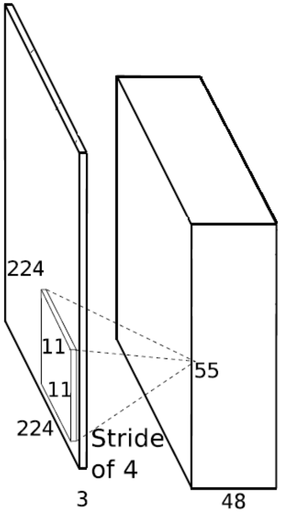
Convolutions



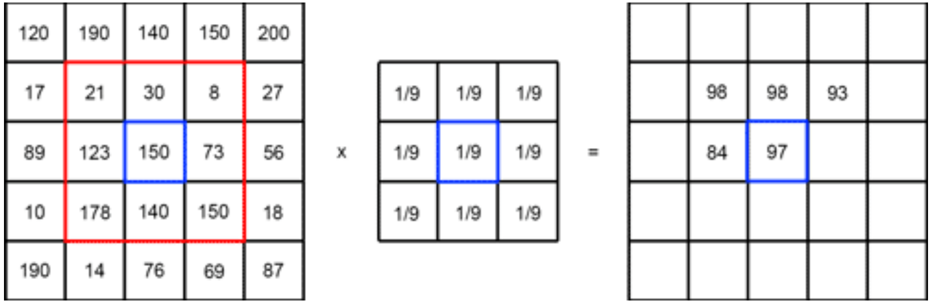


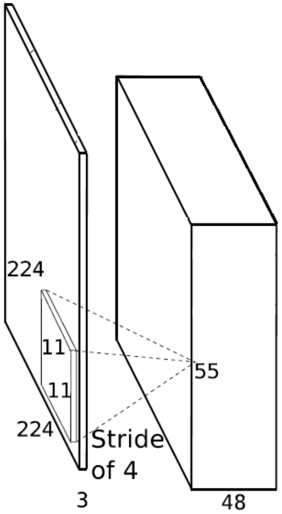
Convolutions



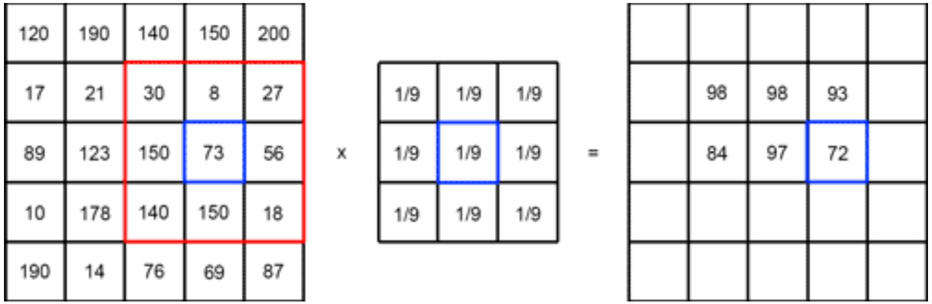


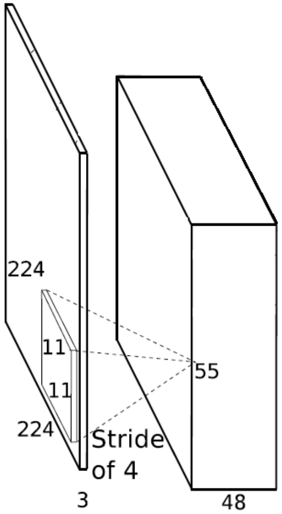
Convolutions



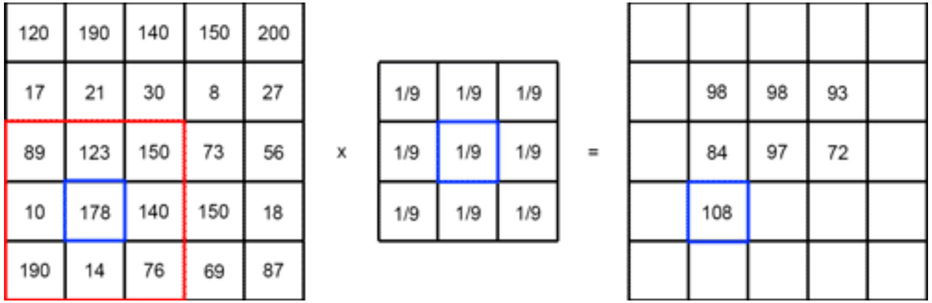


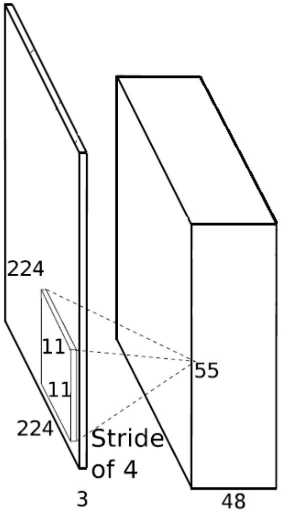
Convolutions





Convolutions





Convolutions

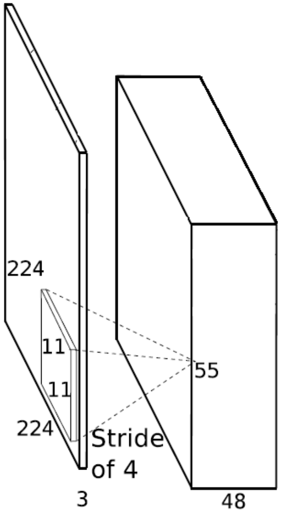
120	190	140	150	200
17	21	30	8	27
89	123	150	73	56
10	178	140	150	18
190	14	76	69	87

x

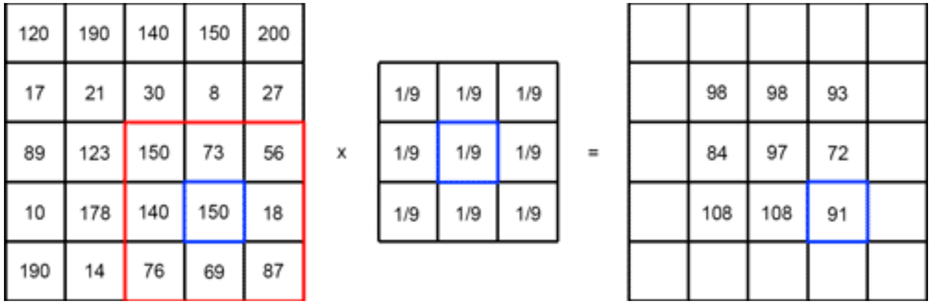
1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

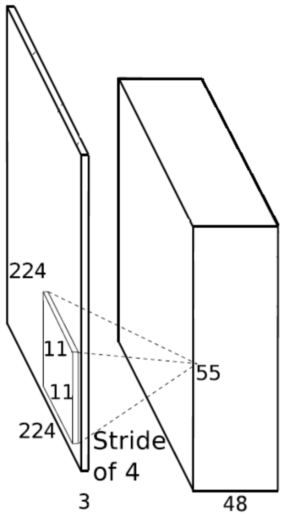
=

	98	98	93	
	84	97	72	
	108	108		

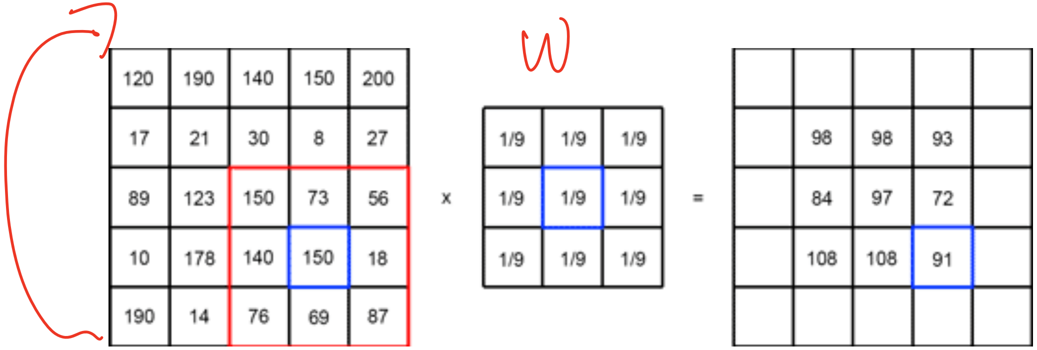


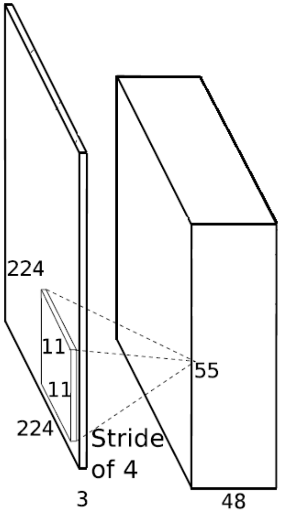
Convolutions





Convolutions





Trainable parameters w:

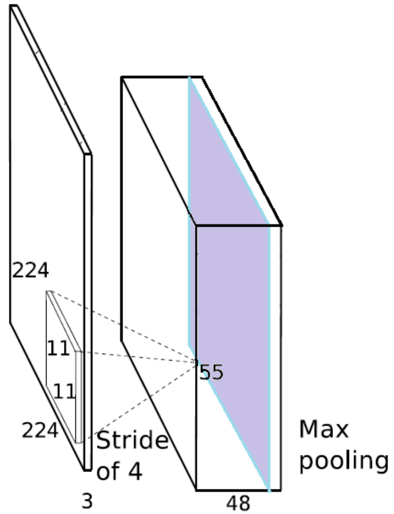
- Filters (width, height, depth, number)
- Bias

Maximum-/Average- pooling

Maximum or average over a spatial region

Trainable parameters w:

- None



Soft-max layer

$$\frac{\exp z_i}{\sum_j \exp z_j}$$

Trainable parameters $\boldsymbol{w}$:

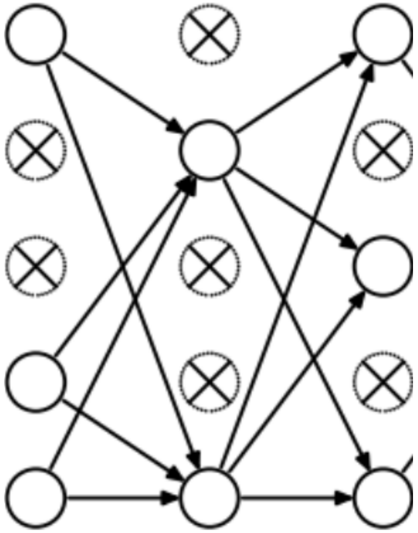
- None

Dropout layer

Randomly set activations to zero

Trainable parameters w:

- None



Multiclass output.

Modern networks often end with **softmax** nonlinearity:

$$\sum_{i=1}^{k} \frac{\exp(\boldsymbol{z}_i)\mathbf{e}_i}{\sum_{j=1}^{k}\exp(\boldsymbol{z}_j)}$$

(where $\mathbf{e}_i$ is i^{th} standard basis vector.)
Output is now a probability vector!

Multiclass output.

Modern networks often end with **softmax** nonlinearity:

$$\sum_{i=1}^{k} \frac{\exp(\boldsymbol{z}_i)\mathbf{e}_i}{\sum_{j=1}^{k} \exp(\boldsymbol{z}_j)}$$

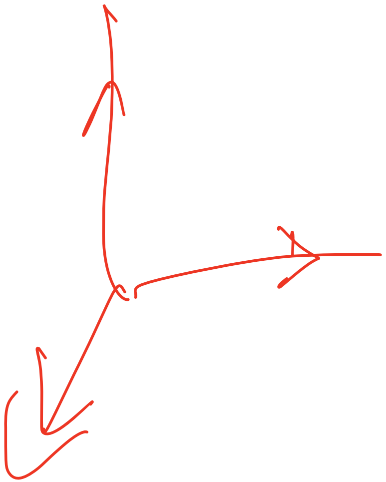
(where $\mathbf{e}_i$ is i^{th} standard basis vector.)
Output is now a probability vector!

Alternate notation: output vector $\boldsymbol{v}_i \propto \exp(\boldsymbol{z}_i)$.

Cross-entropy loss.

Given *one hot* $\boldsymbol{y} \in \{\mathbf{e}_1, \ldots, \mathbf{e}_k\}$ and probability vector $\hat{\boldsymbol{y}} \in \mathbb{R}^k$,

$$\ell(y, \hat{y}) = -\sum_{i=1}^{k} \boldsymbol{y}_i \ln(\hat{\boldsymbol{y}}_i).$$



$$\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ \vdots \end{pmatrix} \quad \begin{pmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{pmatrix} \quad \cdots \quad \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}$$

Cross-entropy loss.

Given *one hot* $\boldsymbol{y} \in \{\mathbf{e}_1, \ldots, \mathbf{e}_k\}$ and probability vector $\hat{\boldsymbol{y}} \in \mathbb{R}^k$,

$$\ell(y, \hat{y}) = -\sum_{i=1}^{k} \boldsymbol{y}_i \ln(\hat{\boldsymbol{y}}_i).$$

Combined with softmax $\hat{\boldsymbol{y}} \propto \exp(\boldsymbol{z})$:

$$-\sum_{i=1}^{k} \boldsymbol{y}_i \ln\left(\frac{\exp(\boldsymbol{z}_i)}{\sum_j \exp(\boldsymbol{z}_j)}\right) = -\sum_{i=1}^{k} \boldsymbol{y}_i \boldsymbol{z}_i + \ln\left(\sum_{i=1}^{k} \exp(\boldsymbol{z}_i)\right).$$

Note: For numerical stability, use $\ln \sum_i \exp v_i = c + \ln \sum_i \exp(v_i - c)$.

Cross-entropy loss.

Given *one hot* $\boldsymbol{y} \in \{\mathbf{e}_1, \ldots, \mathbf{e}_k\}$ and probability vector $\hat{\boldsymbol{y}} \in \mathbb{R}^k$,

$$\ell(y, \hat{y}) = -\sum_{i=1}^{k} \boldsymbol{y}_i \ln(\hat{\boldsymbol{y}}_i).$$

Combined with softmax $\hat{\boldsymbol{y}} \propto \exp(\boldsymbol{z})$:

$$-\sum_{i=1}^{k} \boldsymbol{y}_i \ln\left(\frac{\exp(\boldsymbol{z}_i)}{\sum_j \exp(\boldsymbol{z}_j)}\right) = -\sum_{i=1}^{k} \boldsymbol{y}_i \boldsymbol{z}_i + \ln\left(\sum_{i=1}^{k} \exp(\boldsymbol{z}_i)\right).$$

Note: For numerical stability, use $\ln \sum_i \exp v_i = c + \ln \sum_i \exp(v_i - c)$.

Equivalent to multinomial logistic loss.

Regularization.

- "Weight decay": $+\lambda\|\boldsymbol{w}\|^2$ in objective (where $\boldsymbol{w}$ are all parameters).
- Dropout: randomly nullify node outputs in training.
- Batch normalization: "standardize" node output distribution.

Optimization.

Regularized learning now takes the form

$$\min_{\boldsymbol{W}^{[1]},\ldots,\boldsymbol{W}^{[L]},\boldsymbol{b}^{[1]},\ldots,\boldsymbol{b}^{[L]}} \frac{1}{n}\sum_{i=1}^{n} \ell\left(y^{(i)}, \boldsymbol{W}^{[L]}\sigma_{L-1}\left(\cdots\sigma_1\left(\boldsymbol{W}^{[1]}\boldsymbol{x}^{(i)}+\boldsymbol{b}^{[1]}\right)\cdots\right)\right) + \lambda\sum_{l=1}^{L}\|\boldsymbol{W}^{[l]}\|_2^2$$

Optimization.

Regularized learning now takes the form

$$\min_{\boldsymbol{W}^{[1]},\ldots,\boldsymbol{W}^{[L]},\boldsymbol{b}^{[1]},\ldots,\boldsymbol{b}^{[L]}} \frac{1}{n}\sum_{i=1}^{n} \ell\left(y^{(i)}, \boldsymbol{W}^{[L]}\sigma_{L-1}\left(\cdots\sigma_1\left(\boldsymbol{W}^{[1]}\boldsymbol{x}^{(i)}+\boldsymbol{b}^{[1]}\right)\cdots\right)\right) + \lambda\sum_{l=1}^{L}\|\boldsymbol{W}^{[l]}\|_2^2$$

In general, resulting optimization is "harder" than linear regression

Optimization.

Regularized learning now takes the form

$$\min_{\boldsymbol{W}^{[1]},\ldots,\boldsymbol{W}^{[L]},\boldsymbol{b}^{[1]},\ldots,\boldsymbol{b}^{[L]}} \frac{1}{n}\sum_{i=1}^{n} \ell\left(y^{(i)}, \boldsymbol{W}^{[L]}\sigma_{L-1}\left(\cdots\sigma_1\left(\boldsymbol{W}^{[1]}\boldsymbol{x}^{(i)}+\boldsymbol{b}^{[1]}\right)\cdots\right)\right) + \lambda\sum_{l=1}^{L}\|\boldsymbol{W}^{[l]}\|_2^2$$

In general, resulting optimization is "harder" than linear regression

Implications:

- Gradient-based optimization approaches is no longer guaranteed to find the global optimum
- Initialization of parameters matters

Optimization.

Regularized learning now takes the form

$$\min_{\boldsymbol{W}^{[1]},\ldots,\boldsymbol{W}^{[L]},\boldsymbol{b}^{[1]},\ldots,\boldsymbol{b}^{[L]}} \frac{1}{n}\sum_{i=1}^{n} \ell\left(y^{(i)}, \boldsymbol{W}^{[L]}\sigma_{L-1}\left(\cdots\sigma_1(\boldsymbol{W}^{[1]}\boldsymbol{x}^{(i)}+\boldsymbol{b}^{[1]})\cdots\right)\right) + \lambda\sum_{l=1}^{L}\|\boldsymbol{W}^{[l]}\|_2^2$$

In general, resulting optimization is "harder" than linear regression

Implications:

- Gradient-based optimization approaches is no longer guaranteed to find the global optimum
- Initialization of parameters matters
- Stochastic gradient descent works well in practice

Quiz:

Quiz:

- What are deep neural networks?

Quiz:

- What are deep neural networks?
- How can you interpret deep nets as learning feature transformations?

Quiz:

- What are deep neural networks?
- How can you interpret deep nets as learning feature transformations?
- How can you interpret deep nets as function graphs?

Quiz:

- What are deep neural networks?
- How can you interpret deep nets as learning feature transformations?
- How can you interpret deep nets as function graphs?
- What are some standard components of deep nets?

Quiz:

- What are deep neural networks?
- How can you interpret deep nets as learning feature transformations?
- How can you interpret deep nets as function graphs?
- What are some standard components of deep nets?
- What algorithm can be used to train deep nets?

Important topics of this lecture

- Deep nets as functions and graphs
- Components of deep nets

Up next:

- Backpropagation