Introduction to Neural Networks

Madhavan Mukund

https://www.cmi.ac.in/~madhavan

Focus Programme in Formal Methods and Artificial Intelligenc
IRL ReLaX
2 February 2026

Supervised learning

- A set of items
  - Each item is characterized by attributes $(a_1, a_2, \ldots, a_k)$
  - Each item is assigned a class or category c
- Given a set of examples, predict c for a new item with attributes $(a'_1, a'_2, \ldots, a'_k)$
- Examples provided are called training data
- Aim is to learn a mathematical model that generalizes the training data
  - Model built from training data should extend to previously unseen inputs
- Classification problem
  - Usually assumed to binary — two classes

Example: Loan application data set

ID	Age	Has_job	Own_house	Credit_rating	Class
1	young	false	false	fair	**No**
2	young	false	false	good	**No**
3	young	true	false	good	**Yes**
4	young	true	true	fair	**Yes**
5	young	false	false	fair	**No**
6	middle	false	false	fair	**No**
7	middle	false	false	good	**No**
8	middle	true	true	good	**Yes**
9	middle	false	true	excellent	**Yes**
10	middle	false	true	excellent	**Yes**
11	old	false	true	excellent	**Yes**
12	old	false	true	good	**Yes**
13	old	true	false	good	**Yes**
14	old	true	false	excellent	**Yes**
15	old	false	false	fair	**No**

Basic assumptions

Fundamental assumption of machine learning

- Distribution of training examples is identical to distribution of unseen data

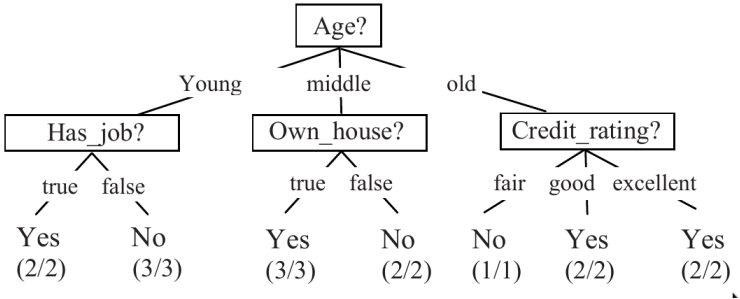
What does it mean to learn from the data?

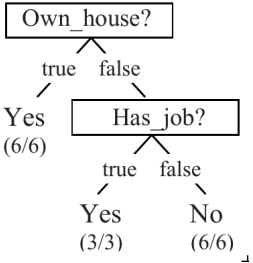
- Build a model that does better than random guessing
  - In the loan data set, always saying Yes would be correct about $9/15$ of the time
- Performance should ideally improve with more training data

How do we evaluate the performance of a model?

- Model is optimized for the training data. How well does it work for unseen data?
- Don't know the correct answers in advance to compare — different from normal software verification

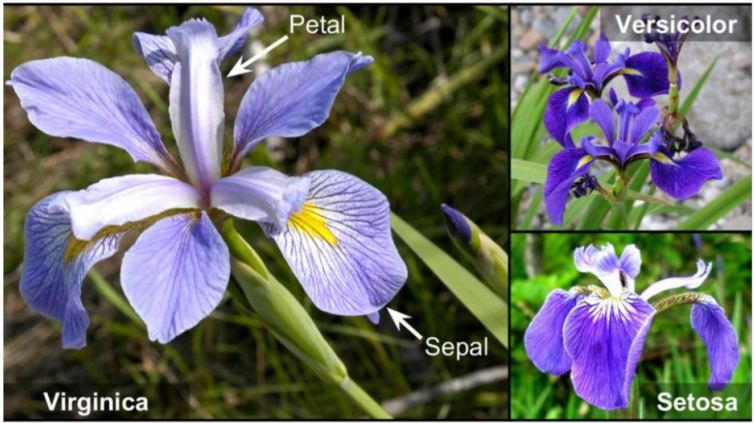
Decision trees





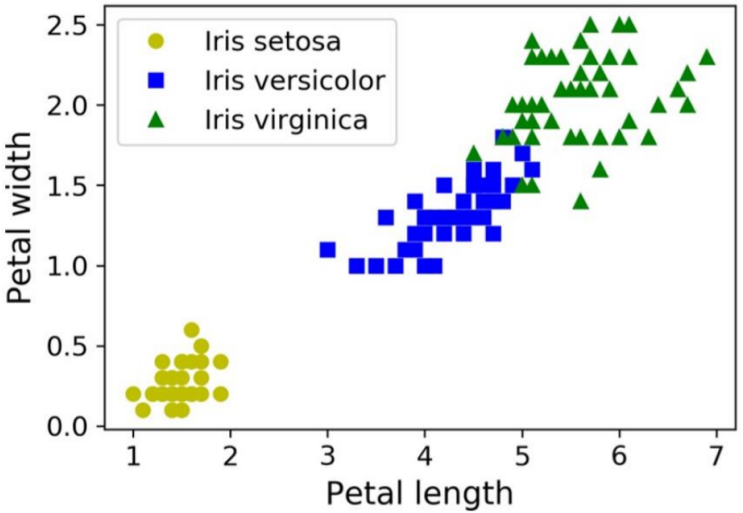
Iris dataset

- Iris is a type of flower
- Three species: *iris setosa*, *iris versicolor*, *iris virginica*
- Dataset has sepal length and width and petal length and width for 150 flowers



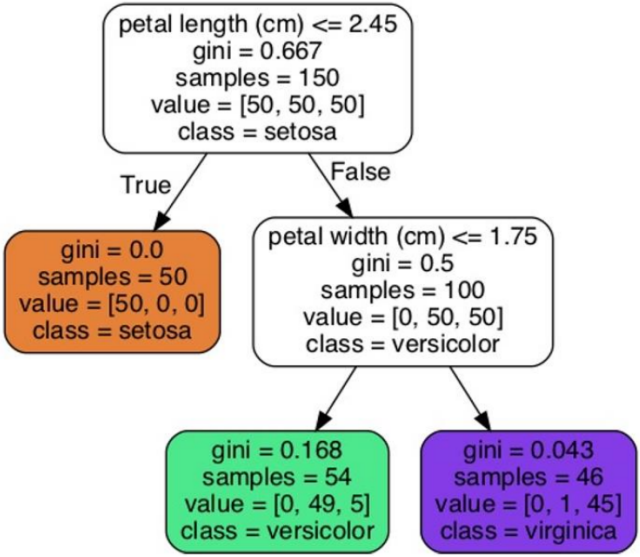
Iris dataset

- Iris is a type of flower
- Three species: *iris setosa*, *iris versicolor*, *iris virginica*
- Dataset has sepal length and width and petal length and width for 150 flowers
- Scatter plot for two attributes, petal length and petal width



Iris dataset

- Iris is a type of flower
- Three species: *iris setosa*, *iris versicolor*, *iris virginica*
- Dataset has sepal length and width and petal length and width for 150 flowers
- Scatter plot for two attributes, petal length and petal width
- Decision tree for this data set



Testing a supervised learning model

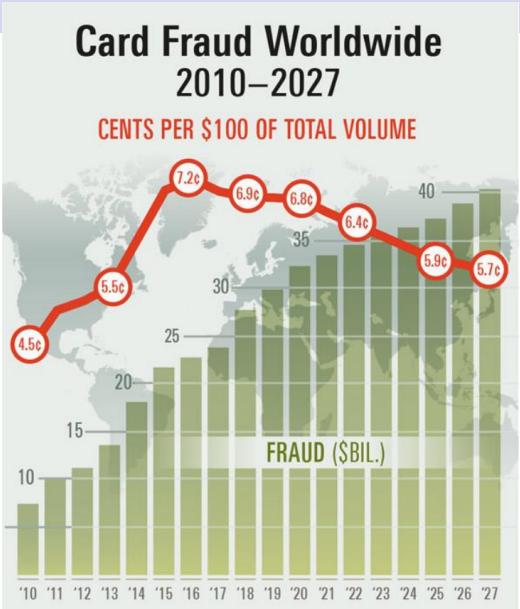
- How do we validate software?
  - Test suite of carefully selected inputs
  - Compare output with expected answers
- What about classification models?
  - By definition, deploy on data where the outcome is unknown
  - If expected answer available, have a deterministic solution, model not needed!
- On what basis can we evaluate a supervised learning model?

Creating a test set

- Training data is labelled
  - No other source of inputs with expected answers
- Segregate some training data for testing
  - Terminology: training set and test set
  - Build model using training set, evaluate on test set
- Creating the test set
  - Need to choose a random sample
  - Can further use stratified sampling, preserve relative ratios (e.g., age wise distribution)
  - ML libraries can do this automatically

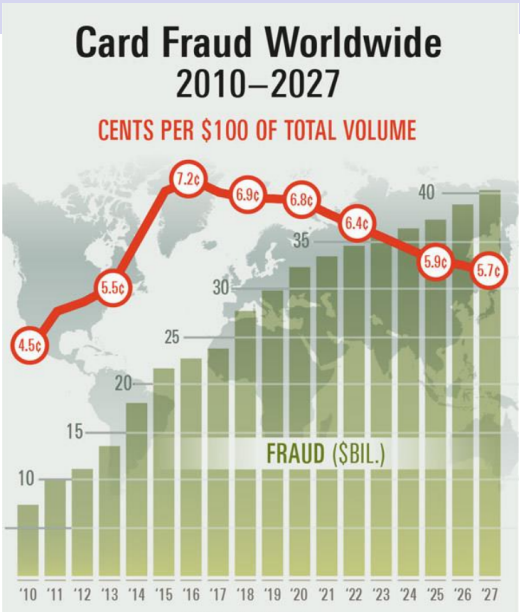
What are we measuring?

- Accuracy is an obvious measure
  - Fraction of inputs where classification is correct
- Classifiers are often used in asymmetric situations
  - Less than 1% of credit card transactions are fraud
- "Is this transaction a fraud?"
  - Trivial classifier — always answer "No"
  - More than 99% accurate, but useless!



Catching the minority case

- The minority case is the useful case
  - Assume question is phrased so that minority answer is "Yes"
  - Want to flag as many "Yes" cases as possible
- Aggressive classifier
  - Marks borderline "No" as "Yes"
  - False positives
- Cautious classifier
  - Marks borderline "Yes" as "No"
  - False negatives



Confusion matrix

- Four possible combinations
  - Actual answer: Yes / No
  - Prediction: Yes / No
- Record all four possibilities in confusion matrix
  - Correct answers
    - True positives, true negatives
  - Wrong answers
    - False positives, false negatives

	Classified positive	Classified negative
Actual positive	True Positive (TP)	False Negative (FN)
Actual negative	False Positive (FP)	True Negative (TN)

Performance measures

Precision

- What percentage of positive predictions are correct?

$$\frac{\text{TP}}{\text{TP} + \text{FP}}$$

Recall

- What percentage of actual positive cases are discovered?

$$\frac{\text{TP}}{\text{TP} + \text{FN}}$$

	Classified positive	Classified negative
Actual positive	True Positive (TP)	False Negative (FN)
Actual negative	False Positive (FP)	True Negative (TN)

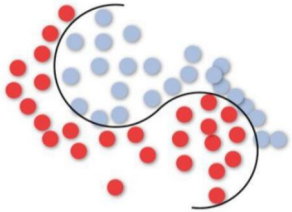
Performance measures

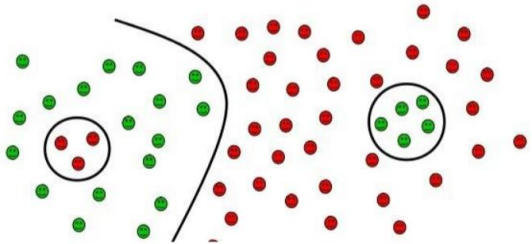
- Precision 1, Recall 0.01
- Recall up to 0.4, but precision down to 0.29
- Recall up to 0.99, but precision down to 0.165
- Precision-recall tradeoff
  - Strict classifiers : fewer false positives (high precision), miss more actual positives (low recall)
  - Permissive classifiers : catch more actual positives (high recall) but more false positives (low precision)

	Classified positive	Classified negative
Actual positive	1 40 99	99 60 1
Actual negative	0 100 500	900 800 400

A geometric view of supervised learning

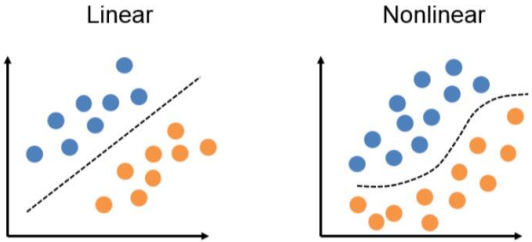
- Think of data as points in space
- Find a separating curve (surface)
- Separable case
  - Each class is a connected region
  - A single curve can separate them
- More complex scenario
  - Classes form multiple connected regions
  - Need multiple separators

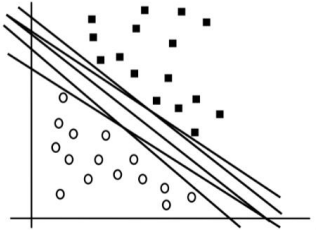




Linear separators

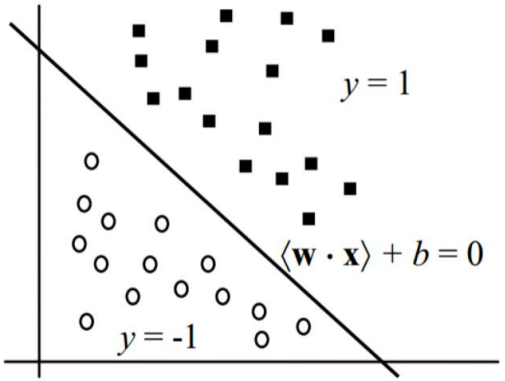
- Simplest case — linearly separable data
- Many lines are possible
  - How do we find the best one?
  - What is a good notion of "cost" to optimize?





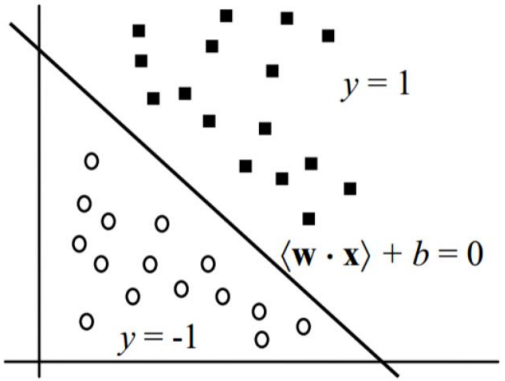
Linear separators

- Each input x has n attributes $\langle x_1, x_2, \ldots, x_n \rangle$
- Linear separator has the form $w_1 x_1 + w_2 x_2 + \cdots w_n x_n + b$
- Classification criterion
  - $w_1 x_1 + w_2 x_2 + \cdots w_n x_n + b > 0$, classify yes, $+1$
  - $w_1 x_1 + w_2 x_2 + \cdots w_n x_n + b < 0$, classify no, -1



Linear separators

- Dot product $w \cdot x$
  $\langle w_1, w_2, \ldots, w_n \rangle \cdot \langle x_1, x_2, \ldots, x_n \rangle = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$
- Collapsed form
  $w \cdot x + b > 0$, $w \cdot x + b < 0$
- Rename bias b as w_0, create fictitious $x_0 = 1$
- Classification criteria become
  $w \cdot x > 0$, $w \cdot x < 0$



Perceptron algorithm

(Frank Rosenblatt, 1958)

- Each training input is (x_i, y_i), where $x_i = \langle x_{i_1}, x_{i_2}, \ldots, x_{i_n} \rangle$ and $y_i = +1$ or -1
- Need to find $w = \langle w_0, w_1, \ldots, w_n \rangle$
  - Recall $x_{i_0} = 1$, always

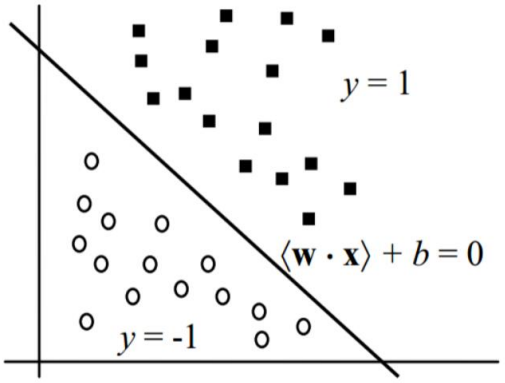
Initialize $w = \langle 0, 0, \ldots, 0 \rangle$

While there exists x_i, y_i *such that*

$y_i = +1$ *and* $w \cdot x_i < 0$, *or*

$y_i = -1$ *and* $w \cdot x_i > 0$

Update w *to* $w + x_i y_i$

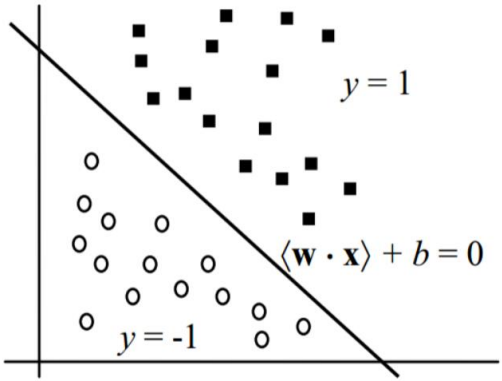


Perceptron algorithm . . .

- Keep updating w as long as some training data item is misclassified
- Update is an offset by misclassified input
- Need not stabilize, potentially an infinite loop

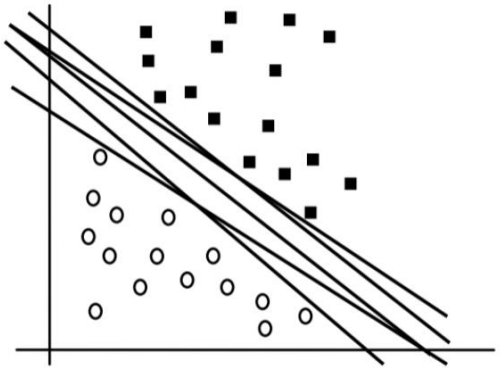
Theorem

If the points are linearly separable, the Perceptron algorithms always terminates with a valid separator



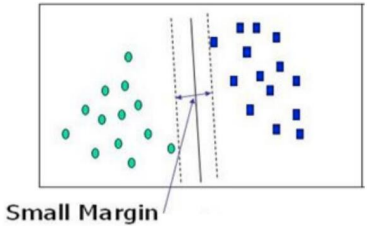
Linear separators

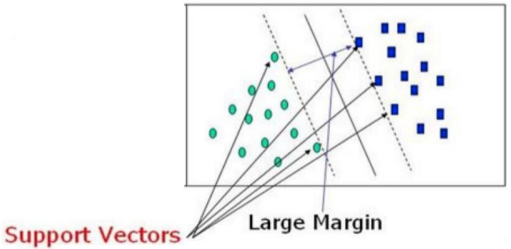
- Simplest case — linearly separable data
- Perceptron algorithm is a simple procedure to find a linear separator, if one exists
- Many lines are possible
  - Does the Perceptron algorithm find the best one?
  - What is a good notion of "cost" to optimize?



Margin

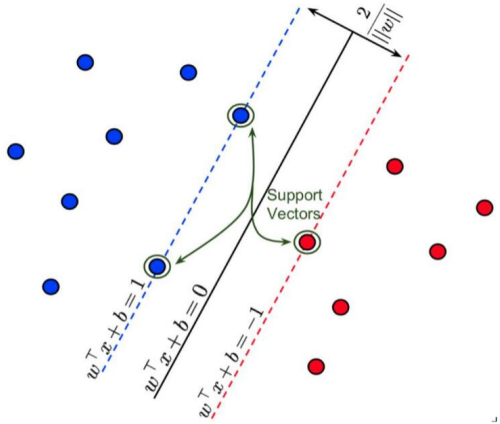
- Each separator defines a margin
  - Empty corridor separating the points
  - Separator is the centre line of the margin
- Wider margin makes for a more robust classifier
  - More gap between the classes
- Optimum classifier is one that maximizes the width of its margin
- Margin is defined by the training data points on the boundary
  - Support vectors





Finding a maximum margin classifier

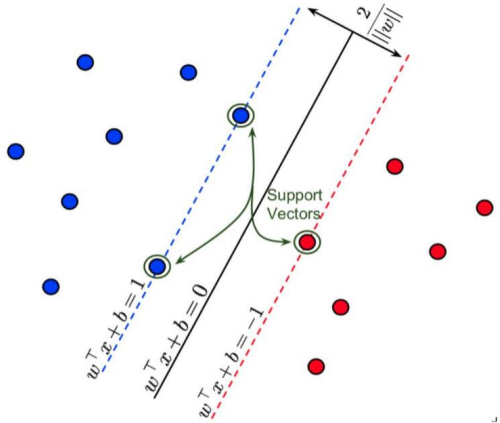
- Recall our original linear classifier
  - $w_1x_1 + w_2x_2 + \cdots w_nx_n + b > 0$, classify yes, $+1$
  - $w_1x_1 + w_2x_2 + \cdots w_nx_n + b < 0$, classify no, -1
- Scale margin so that separation is 1 on either side
  - $w_1x_1 + w_2x_2 + \cdots w_nx_n + b > 1$, classify yes, $+1$
  - $w_1x_1 + w_2x_2 + \cdots w_nx_n + b < -1$, classify no, -1



Finding a maximum margin classifier

- Scale margin so that separation is 1 on either side
  - $w_1x_1 + w_2x_2 + \cdots w_nx_n + b > 1$, classify yes, $+1$
  - $w_1x_1 + w_2x_2 + \cdots w_nx_n + b < -1$, classify no, -1
- Using Pythagoras's theorem, perpendicular distance to nearest support vector is $\frac{1}{|w|}$, where

$$|w| = \sqrt{w_1^2 + w_2^2 + \cdots + w_n^2}$$

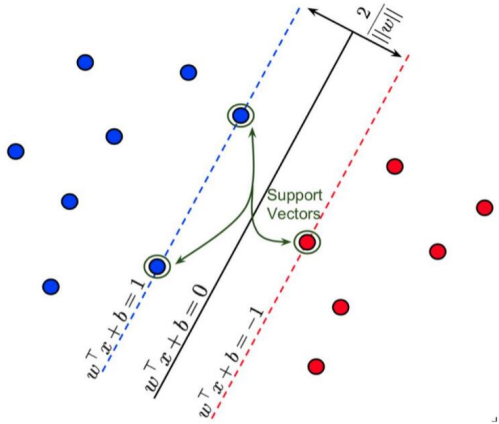


Optimization problem

- Want to maximize the overall margin $\frac{2}{|w|}$
- Equivalently, minimize $\frac{|w|}{2}$
- Also, w should classify each (x_i, y_i) correctly

  $w_1 x_1^i + w_2 x_2^i + \cdots w_n x_n^i + b > 1$, if $y_i = 1$

  $w_1 x_1^i + w_2 x_2^i + \cdots w_n x_n^i + b < -1$, if $y_i = -1$



Optimization problem

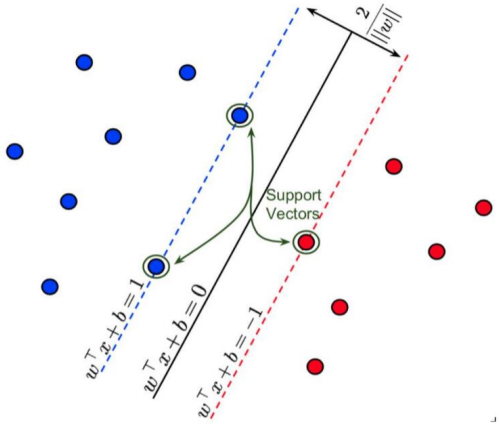
Minimize $\frac{|w|}{2}$

Subject to

$w_1 x_1^i + w_2 x_2^i + \cdots w_n x_n^i + b > 1$, if $y_i = 1$

$w_1 x_1^i + w_2 x_2^i + \cdots w_n x_n^i + b < -1$, if $y_i = -1$

- The constraints are linear
- The objective function is not linear
  $|w| = \sqrt{w_1^2 + w_2^2 + \cdots + w_n^2}$
- This is a quadratic optimization problem, not linear programming

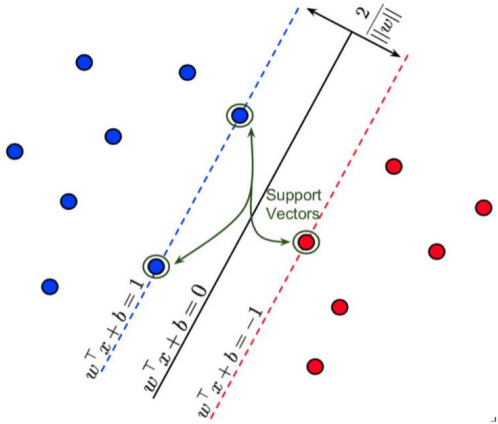


Solution to optimization problem

- Convex optimization theory
- Can be solved using computational techniques
- Solution expressed in terms of Lagrange multipliers $\alpha_1, \alpha_2, \ldots, \alpha_N$
- α_i is non-zero iff x_i is a support vector
- Final classifier for new input z

$$\text{sign}\left[\sum_{i \in sv} y_i \alpha_i (x_i \cdot z) + b\right]$$

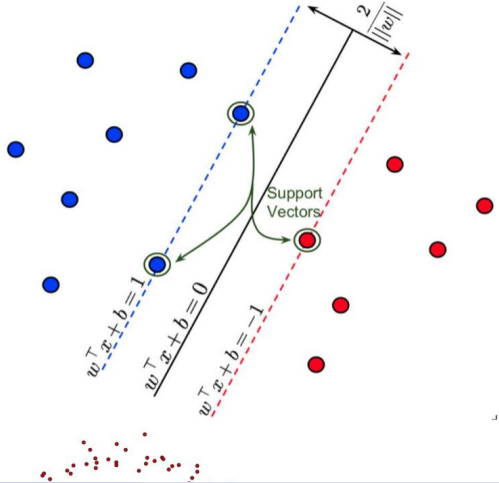
- sv is set of support vectors



Support Vector Machine (SVM)

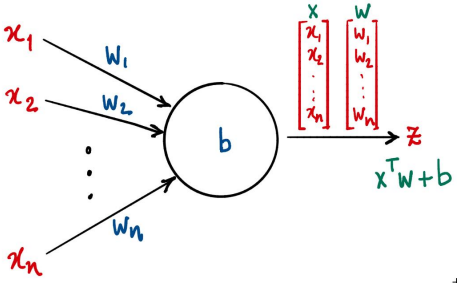
$$\text{sign}\left[\sum_{i \in sv} y_i \alpha_i (x_i \cdot z) + b\right]$$

- Solution depends only on support vectors
  - If we add more training data away from support vectors, separator does not change
- Data not linearly separable?
  - Geometric transformation to a higher dimension
- Kernel methods



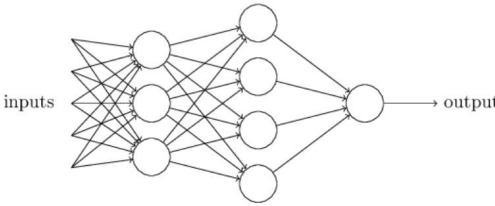
Linear separators and Perceptrons

- Perceptrons define linear separators $w \cdot x + b$
  - $w \cdot x + b > 0$, classify Yes $(+1)$
  - $w \cdot x + b < 0$, classify No (-1)



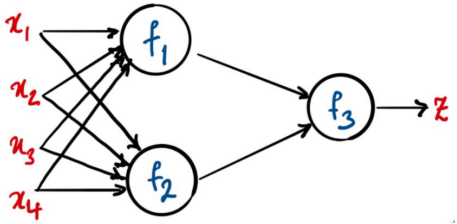
Linear separators and Perceptrons

- Perceptrons define linear separators $w \cdot x + b$
  - $w \cdot x + b > 0$, classify Yes $(+1)$
  - $w \cdot x + b < 0$, classify No (-1)
- What if we cascade perceptrons?
- Result is still a linear separator



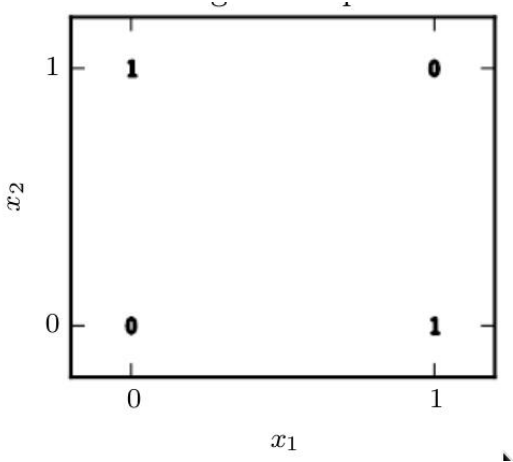
Linear separators and Perceptrons

- Perceptrons define linear separators $w \cdot x + b$
  - $w \cdot x + b > 0$, classify Yes $(+1)$
  - $w \cdot x + b < 0$, classify No (-1)
- What if we cascade perceptrons?
- Result is still a linear separator
  - $f_1 = w_1 \cdot x + b_1, f_2 = w_2 \cdot x + b_2$
  - $f_3 = w_3 \cdot \langle f_1, f_2 \rangle + b_3$
  - $f_3 = w_3 \cdot \langle w_1 \cdot x + b_1, w_2 \cdot x + b_2 \rangle + b_3$
  - $f_3 = \sum_{i=1}^{4} (w_{3_1} w_{1_i} + w_{3_2} w_{2_i}) \cdot x_i$
    $+(w_{3_1} b_1 + w_{3_2} b_2 + b_3)$



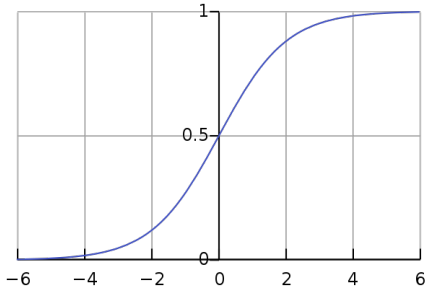
Limits of linearity

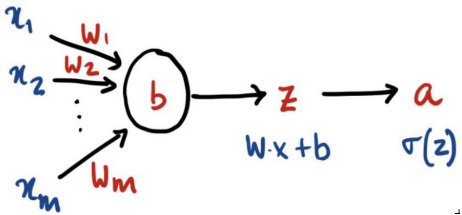
- Cannot compute *exclusive-or* (XOR)
- $XOR(x_1, x_2)$ is true if exactly one of x_1, x_2 is true (not both)
- Suppose $XOR(x_1, x_2) = ux_1 + vx_2 + b$
- $x_2 = 0$: As x_1 goes from 0 to 1, output goes from 0 to 1, so $u > 0$
- $x_2 = 1$: As x_1 goes from 0 to 1, output goes from 1 to 0, so $u < 0$
- Observed by Minsky and Papert, 1969, first "AI Winter"



Non-linear activation

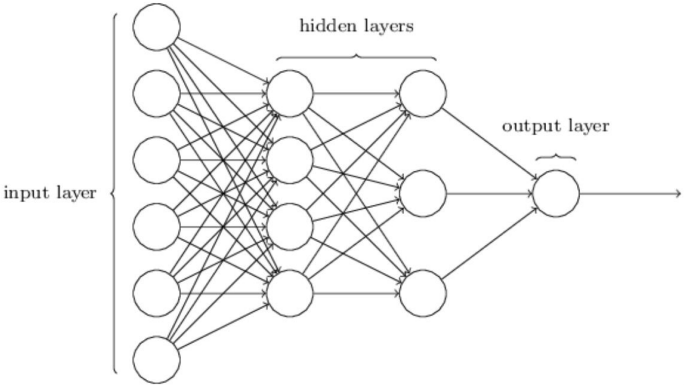
- Transform linear output z through a non-linear activation function
- Sigmoid function $\dfrac{1}{1+e^{-z}}$





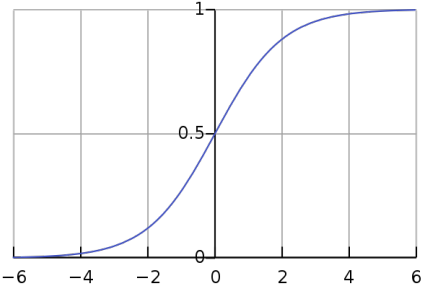
Structure of a neural network

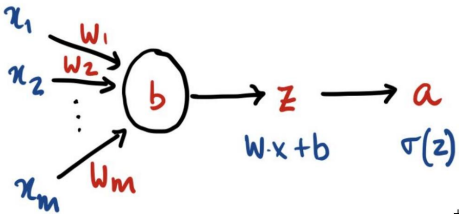
- Acyclic
- Input layer, hidden layers, output layer
- Assumptions
  - Hidden neurons are arranged in layers
  - Each layer is fully connected to the next
  - Set weight to zero to remove an edge



Non-linear activation

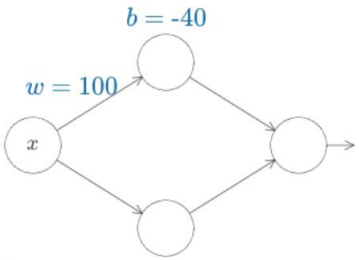
- Transform linear output z through a non-linear activation function
- Sigmoid function $\dfrac{1}{1+e^{-z}}$
- Step is at $z = 0$
  - $z = wx + b$, so step is at $x = -b/w$
  - Increasing w makes step steeper
  - Shift step by adjusting b

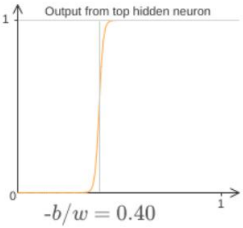




Universality

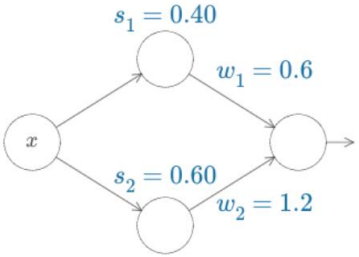
- Create a step at $x = -b/w$
  - Use s to denote $-b/w$
  - Step position

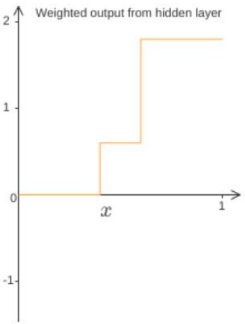




Universality

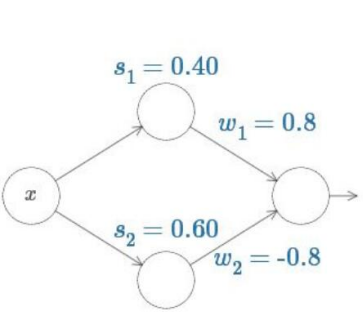
- Create a step at $x = -b/w$
  - Use s to denote $-b/w$
  - Step position
- Cascade steps





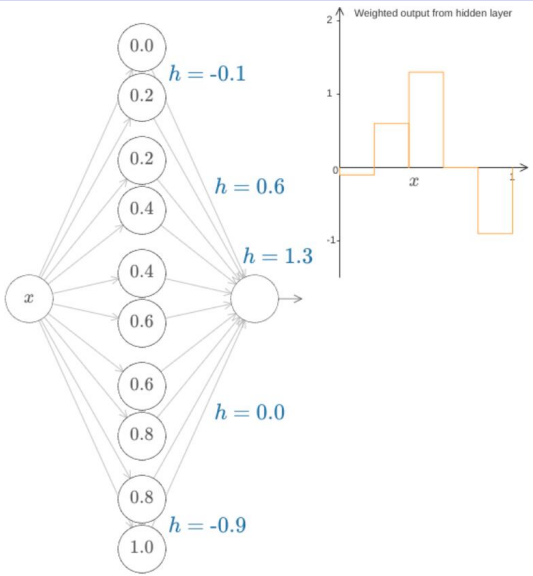
Universality

- Create a step at $x = -b/w$
  - Use s to denote $-b/w$
  - Step position
- Cascade steps
- Subtract steps to create a box



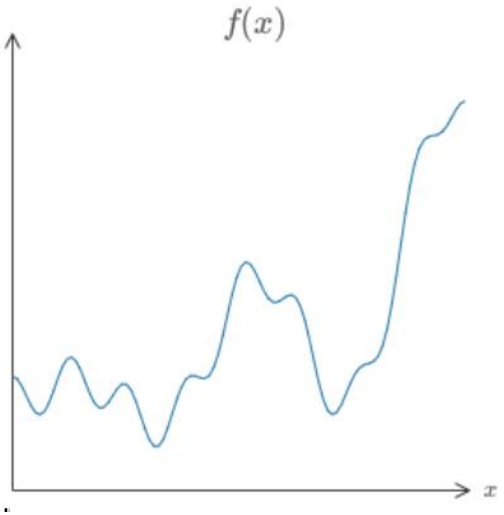
Universality

- Create a step at $x = -b/w$
  - Use s to denote $-b/w$
  - Step position
- Cascade steps
- Subtract steps to create a box
- Create many boxes



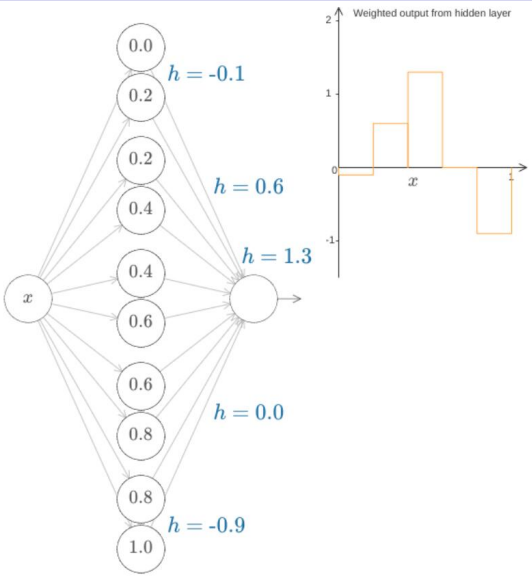
Universality

- Create a step at $x = -b/w$
  - Use s to denote $-b/w$
  - Step position
- Cascade steps
- Subtract steps to create a box
- Create many boxes
- Approximate any function



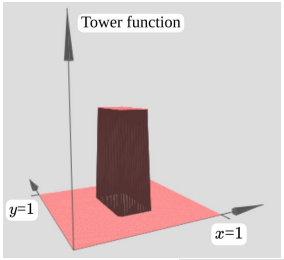
Universality

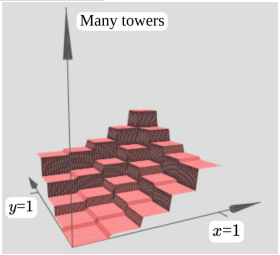
- Create a step at $x = -b/w$
  - Use s to denote $-b/w$
  - Step position
- Cascade steps
- Subtract steps to create a box
- Create many boxes
- Approximate any function
- Need only one hidden layer!



Non-linear activation

- With non-linear activation, network of neurons can approximate any function
  - Can build "rectangular" blocks
  - Combine blocks to capture any classification boundary



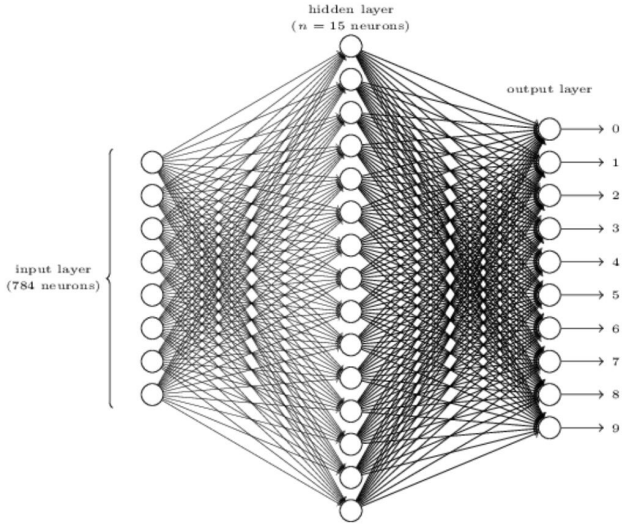


Example: Recognizing handwritten digits

- MNIST data set
- 1000 samples of 10 handwritten digits
  - Assume input has been segmented
- Each digit is 28×28 pixels
  - Grayscale value, 0 to 1
  - 784 pixels
- Input $x = (x_1, x_2, \ldots, x_{784})$

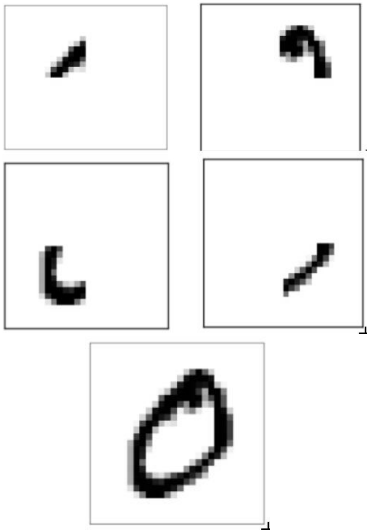
Example: Network structure

- Input layer $(x_1, x_2, \ldots, x_{784})$
- Single hidden layer, 15 nodes
- Output layer, 10 nodes
  - Decision a_j for each digit $j \in \{0, 1, \ldots, 9\}$
- Final output is best a_j
  - Naïvely, $\arg\max_j a_j$
  - Softmax, $\arg\max_j \frac{e^{a_j}}{\sum_j e^{a_j}}$
    - "Smooth" version of arg max



Example: Extracting features

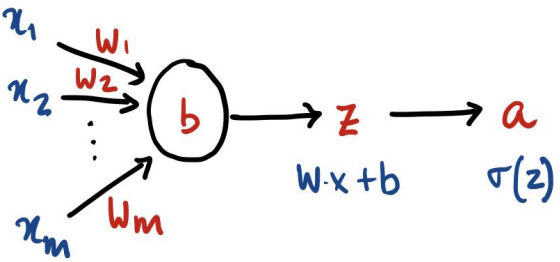
- Hidden layers extract features
  - For instance, patterns in different quadrants
- Combination of features determines output
- Claim: Automatic identification of features is strength of the model
- Counter argument: implicitly extracted features are impossible to interpret
  - Explainability



Training neural networks

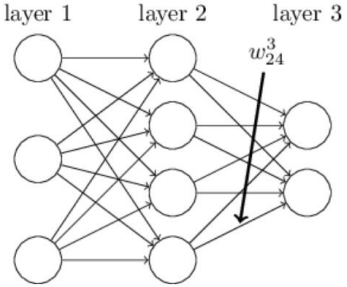
- Without loss of generality,
  - Assume the network is layered
    - All paths from input to output have the same length
  - Each layer is fully connected to the previous one
    - Set weight to 0 if connection is not needed

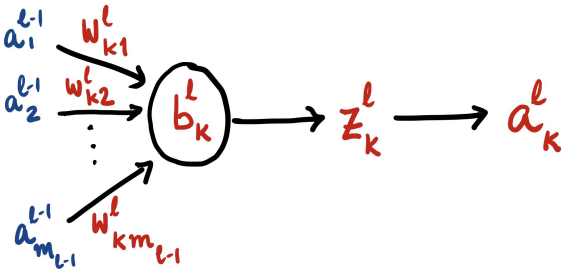
- Structure of an individual neuron
  - Input weights $w_1, \ldots, w_m$, bias b, output z, activation value a



Notation

- Layers $\ell \in \{1, 2, \ldots, L\}$
  - Inputs are connected first hidden layer, layer 1
  - Layer L is the output layer
- Layer ℓ has m_ℓ nodes $1, 2, \ldots, m_\ell$
- Node k in layer ℓ has bias b_k^ℓ, output z_k^ℓ and activation value a_k^ℓ
- Weight on edge from node j in level $\ell-1$ to node k in level ℓ is w_{kj}^ℓ





Notation

- Assume all layers have same number of nodes
  - Let $m = \max_{\ell \in \{1.2,\ldots,L\}} m_\ell$
  - For any layer i, for $k > m_i$, we set all of $w_{kj}^\ell, b_k^\ell, z_k^\ell, a_k^\ell$ to 0
- Matrix formulation

$$\begin{bmatrix} \overline{z}_1^\ell \\ \overline{z}_2^\ell \\ \cdots \\ \overline{z}_m^\ell \end{bmatrix} = \begin{bmatrix} \overline{w}_1^\ell \\ \overline{w}_2^\ell \\ \cdots \\ \overline{w}_m^\ell \end{bmatrix} \begin{bmatrix} a_1^{\ell-1} \\ a_2^{\ell-1} \\ \cdots \\ a_m^{\ell-1} \end{bmatrix}$$

Learning the parameters

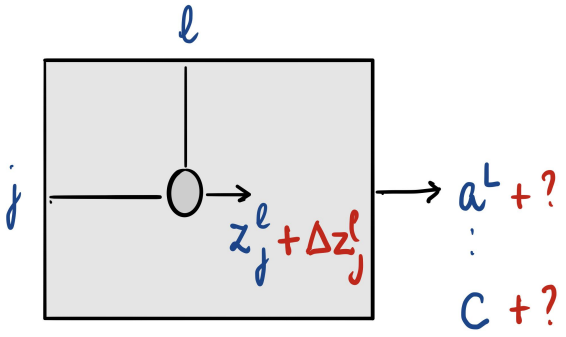
- Need to find optimum values for all weights w^{ℓ}_{kj}
- Use gradient descent
  - Cost function C, partial derivatives $\frac{\partial C}{\partial w^{\ell}_{kj}}$, $\frac{\partial C}{\partial b^{\ell}_{k}}$
- Assumptions about the cost function
  1. For input $\boldsymbol{x}$, $C(\boldsymbol{x})$ is a function of only the output layer activation, a^L
     - For instance, for training input $(\boldsymbol{x}_i, y_i)$, sum-squared error is $(y_i - a^L_i)^2$
     - Note that $\boldsymbol{x}_i$, y_i are fixed values, only a^L_i is a variable
  2. Total cost is average of individual input costs
     - Each input $\boldsymbol{x}_i$ incurs cost $C(\boldsymbol{x}_i)$, total cost is $\frac{1}{n}\sum_{i=1}^{n} C(\boldsymbol{x}_i)$
     - For instance, mean sum-squared error $\frac{1}{n}\sum_{i=1}^{n}(y_i - a^L_i)^2$

Learning the parameters

- Assumptions about the cost function
  1. For input $\boldsymbol{x}$, $C(\boldsymbol{x})$ is a function of only the output layer activation, a^L
  2. Total cost is average of individual input costs
- With these assumptions:
  - We can write $\frac{\partial C}{\partial w_{kj}^\ell}$, $\frac{\partial C}{\partial b_k^\ell}$ in terms of individual $\frac{\partial a_i^L}{\partial w_{kj}^\ell}$, $\frac{\partial a_i^L}{\partial b_k^\ell}$
  - Can extrapolate change in individual cost $C(x)$ to change in overall cost C — stochastic gradient descent
- Complex dependency of C on w_{kj}^ℓ, b_k^ℓ
  - Many intermediate layers
  - Many paths through these layers
- Use chain rule to decompose into local dependencies
  - $y = g(f(x)) \Rightarrow \frac{\partial g}{\partial x} = \frac{\partial g}{\partial f}\frac{\partial f}{\partial x}$

Calculating dependencies

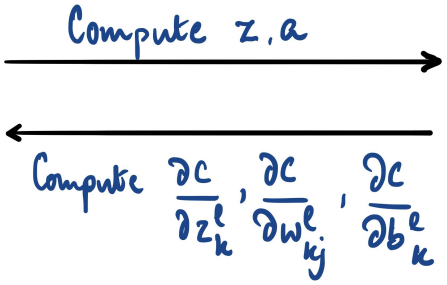
- If we perturb the output z_j^ℓ at node j in layer ℓ, what is the impact on final output, overall cost?



- Focus on $\dfrac{\partial C}{\partial z_j^\ell}$ — from these, we can compute $\dfrac{\partial C}{\partial w_{kj}^\ell}, \dfrac{\partial C}{\partial b_k^\ell}$

Computing partial derivatives

- Use chain rule to run backpropagation algorithm
  - Given an input, execute the network from left to right to compute all outputs
  - Using the chain rule, work backwards from right to left to compute all values of $\frac{\partial C}{\partial z_j^\ell}$



Applying the chain rule

Let δ_j^ℓ denote $\dfrac{\partial C}{\partial z_j^\ell}$

Base Case

$\ell = L$, δ_j^L

- Chain rule: $\dfrac{\partial C}{\partial z_j^L} = \dfrac{\partial C}{\partial a_j^L}\dfrac{\partial a_j^L}{\partial z_j^L}$
- $C = \dfrac{1}{n}\sum_{i=1}^{n}(y_i - a_i^L)^2$, so $\dfrac{\partial C}{\partial a_j^L} = 2(y_j - a_j^L)(-1) = 2(a_j^L - y_j)$
- $a_j^L = \sigma(z_j^L)$, so $\dfrac{\partial a_j^L}{\partial z_j^L} = \sigma'(z_j^L)$
  - $\sigma(u) = \dfrac{1}{1+e^{-u}}$, $\sigma'(u) = \dfrac{\partial \sigma(u)}{\partial u} = \sigma(u)(1-\sigma(u))$ Work this out!

Applying the chain rule

Induction step

From $\delta_j^{\ell+1}$ to δ_j^{ℓ}

- $\delta_j^{\ell} = \dfrac{\partial C}{\partial z_j^{\ell}} = \displaystyle\sum_{k=1}^{m} \frac{\partial C}{\partial z_k^{\ell+1}} \frac{\partial z_k^{\ell+1}}{\partial z_j^{\ell}}$
- First term inside summation: $\dfrac{\partial C}{\partial z_k^{\ell+1}} = \delta_k^{\ell+1}$
- Second term: $z_k^{\ell+1} = \displaystyle\sum_{i=1}^{m} w_{ki}^{\ell+1} a_i^{\ell} + b_k^{\ell+1} = \sum_{i=1}^{m} w_{ki}^{\ell+1} \sigma(z_i^{\ell}) + b_k^{\ell+1}$
  - For $i \neq j$, $\dfrac{\partial}{\partial z_j^{\ell}} [w_{ki}^{\ell+1} \sigma(z_i^{\ell}) + b_k^{\ell+1}] = 0$
  - For $i = j$, $\dfrac{\partial}{\partial z_j^{\ell}} [w_{kj}^{\ell+1} \sigma(z_j^{\ell}) + b_k^{\ell+1}] = w_{kj}^{\ell+1} \sigma'(z_j^{\ell})$
  - So $\dfrac{\partial z_k^{\ell+1}}{\partial z_i^{\ell}} = w_{kj}^{\ell+1} \sigma'(z_j^{\ell})$

Finishing touches

What we actually need to compute are $\frac{\partial C}{\partial w_{kj}^{\ell}}$, $\frac{\partial C}{\partial b_{k}^{\ell}}$

- $\frac{\partial C}{\partial w_{kj}^{\ell}} = \frac{\partial C}{\partial z_{k}^{\ell}} \frac{\partial z_{k}^{\ell}}{\partial w_{kj}^{\ell}} = \delta_{k}^{\ell} \frac{\partial z_{k}^{\ell}}{\partial w_{kj}^{\ell}}$
- $\frac{\partial C}{\partial b_{k}^{\ell}} = \frac{\partial C}{\partial z_{k}^{\ell}} \frac{\partial z_{k}^{\ell}}{\partial b_{k}^{\ell}} = \delta_{k}^{\ell} \frac{\partial z_{k}^{\ell}}{\partial b_{k}^{\ell}}$

We have already computed δ_{k}^{ℓ}, so what remains is $\frac{\partial z_{k}^{\ell}}{\partial w_{kj}^{\ell}}$, $\frac{\partial z_{k}^{\ell}}{\partial b_{k}^{\ell}}$

- Since $z_{k}^{\ell} = \sum_{i=1}^{m} w_{ki}^{\ell} a_{i}^{\ell-1} + b_{k}^{\ell}$, it follows that
  - $\frac{\partial z_{k}^{\ell}}{\partial w_{kj}^{\ell}} = a_{j}^{\ell-1}$ — terms with $i \neq j$ vanish
  - $\frac{\partial z_{k}^{\ell}}{\partial b_{k}^{\ell}} = 1$ — terms with $i \neq j$ vanish

Backpropagation

- In the forward pass, compute all z_k^ℓ, a_k^ℓ
- In the backward pass, compute all δ_k^ℓ, from which we can get all $\frac{\partial C}{\partial w_{kj}^\ell}$, $\frac{\partial C}{\partial b_k^\ell}$
- Increment each parameter by a step Δ in the direction opposite the gradient

Typically, partition the training data into groups (mini batches)

- Update parameters after each mini batch — stochastic gradient descent
- Epoch — one pass through the entire training data

Challenges

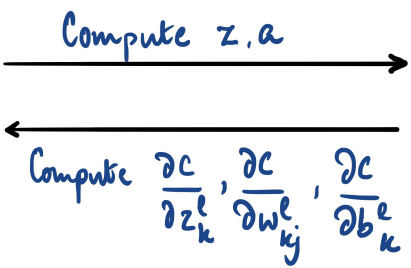
- Backpropagation dates from mid-1980's

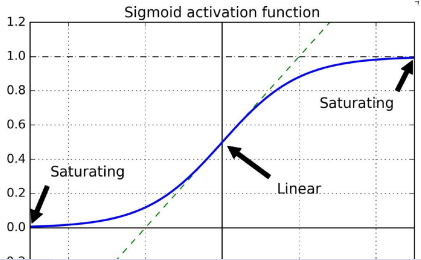
  Learning representations by back-propagating errors
  David E. Rumelhart, Geoffrey E. Hinton and Ronald J. Williams
  Nature, **323**, 533–536 (1986)

- Computationally infeasible till advent of modern parallel hardware, GPUs for vector (tensor) calculations

Unstable gradients

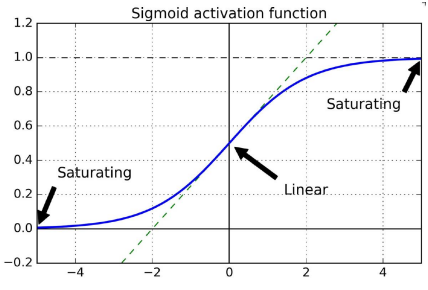
- Vanishing gradients — gradients become smaller towards lower layers (closer to input)
  - Gradient descent updates leave these layers' parameters virtually unchanged
- Also exploding gradients, recurrent neural networks with feedback edges
- In general, unstable gradients, different layers learn at different speeds
- [Xavier Glorot and Joshua Bengio, 2010]
  - Random initialization, traditionally Gaussian distribution $\mathcal{N}(0, 1)$
  - Variance keeps increasing going forward
  - Saturating sigmoid function





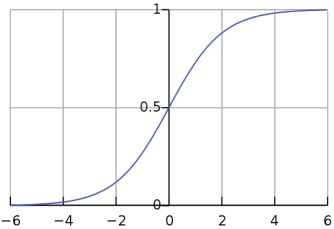
Initializing neural networks

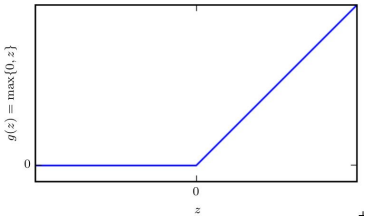
- Want "signal" to flow well in both directions during backpropagation
  - Signal should not die out, explode, saturate
- [Glorot,Bengio] Gradients should have equal variance before and after flowing through a layer in both directions
  - Equal variance requires $fan_{in} = fan_{out}$
- Let $fan_{avg} = (fan_{in} + fan_{out})/2$
- Initialize with
  - Gaussian, $\mathcal{N}(0, 1/fan_{avg})$
  - Uniform, $\mathcal{U}(-r, r)$, $r = \sqrt{\frac{3}{fan_{avg}}}$



Non-saturating activation functions

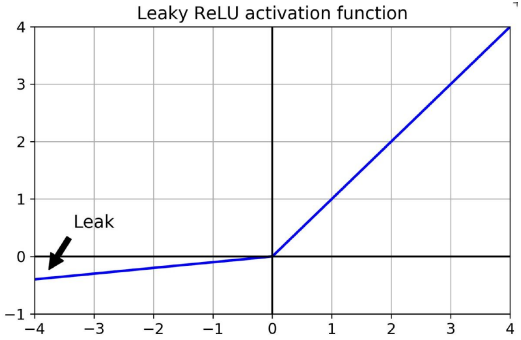
- Sigmoid was initially chosen as a "smooth" step
- Rectified linear unit (ReLU): $g(z) = \max(0, z)$
  - Fast to compute
  - Non-differentiable point not a bottleneck
- "Dying ReLU"
  - Neuron dies — weighted sum of outputs is negative for all training samples
  - With a large learning rate, half the network may die!





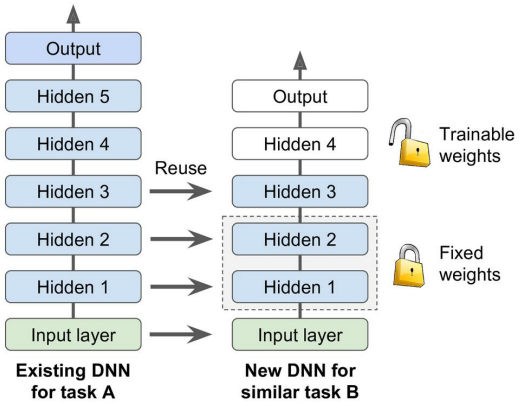
Non-saturating activation functions

- Leaky ReLU, $\max(\alpha z, z)$
  - "Leak" α is a hyperparameter
- RReLU — random leak
  - Pick α from a random range during training
  - Fix to an average value when testing
  - Seems to work well, act as a regularizer



Transfer learning

- Reuse trained layers across deep neural networks (DNNs)
- Old DNN trained on images of daily objects (animals, plants, vehicles, . . .)
- New DNN to classify types of vehicles
- Tasks similar, even overlapping
- Lower layers identify basic features, upper layers combine them to classify
- Freeze weights of lower layers, re-learn upper layers
- Unfreeze in stages to determine how much to reuse

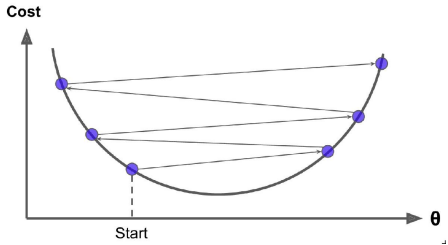


Ill conditioning

- Ill conditioning — small change in input produces a large change in output
- Gradient $\nabla_\theta = \frac{\partial}{\partial \theta_i} J(\theta)$
- Impact of update $\theta - \epsilon \nabla_\theta$ on cost $J(\theta)$?
- Depends on curvature, given by second derivative
- Hessian: $H_\theta = \frac{\delta^2}{\delta \theta_i \delta \theta_j} J(\theta)$
- Using Taylor expansion, impact of update $\theta - \epsilon \nabla_\theta$,

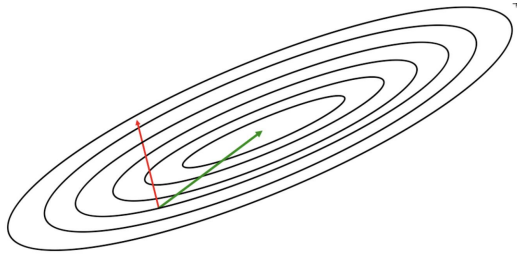
$$J(\theta) - \nabla_\theta^T \nabla_\theta + \frac{1}{2} \nabla_\theta^T H_\theta \nabla_\theta$$

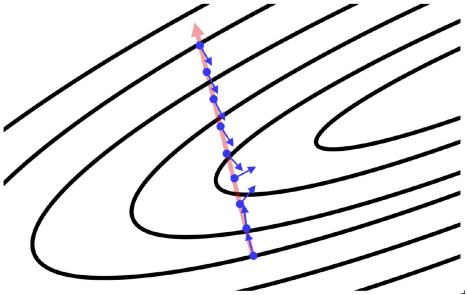
- Analyze H_θ to check for ill conditioning



Directing gradient descent

- Locally steepest direction of descent may be far from the optimum
  - Elliptical contours vs circular contours
- Gradient changes rapidly along the direction of steepest descent
  - Taking large steps is problematic
- Ill-conditioned Hessian H — second derivatives
  - Computing Hessian is expensive
  - "Second order" methods are not used in practice
- Instead, heuristics like momentum and adaptive learning rates



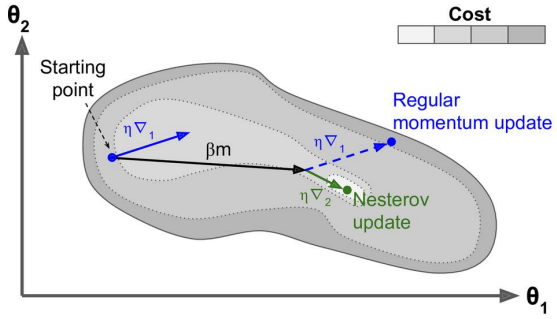


Momentum

- SGD convergence can be very slow
- Momentum in physics — mass $\times$ velocity
- Introduce velocity v in SGD — assume unit mass
  - Moving average of past gradients, exponential decay
  - If gradient remains steady, velocity increases
- Update rule
  - $v \leftarrow \alpha v - \epsilon \nabla_\theta \left(\frac{1}{m} \sum_{i=1}^{m} L(f(x_i; \theta), y_i) \right)$
  - $\theta \leftarrow \theta + v$
- Hyperparameter $\alpha \in [0, 1)$ — "friction", exponentially decaying history
  - With constant gradient g, in the limit $\frac{\epsilon g}{1 - \alpha}$, geometric progression

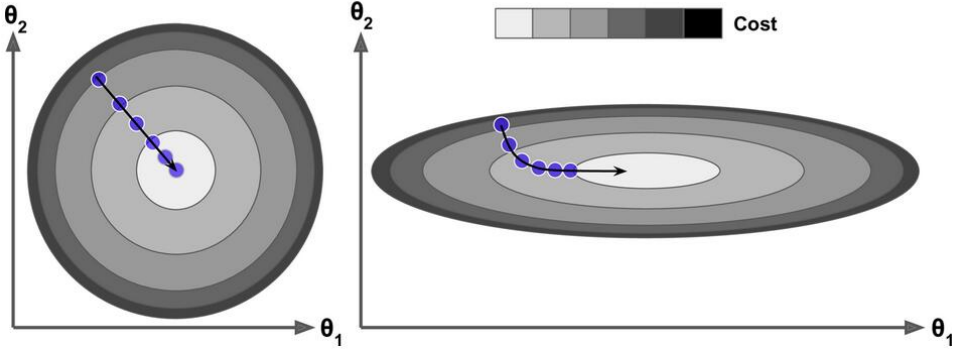
Nesterov momentum optimization

- Measure cost function slightly ahead, in direction of momentum
- Update rule
  - $v \leftarrow \alpha v -$
    $$\epsilon \nabla_\theta \left(\frac{1}{m} \sum_{i=1}^{m} L(f(x_i; \theta + \beta m), y_i) \right)$$
  - $\theta \leftarrow \theta + v$
- Controls sideways oscillations better



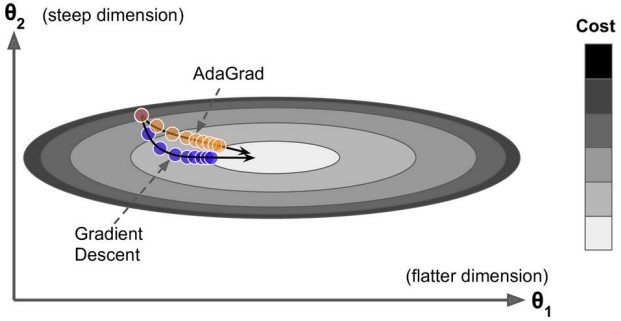
Adjusting the trajectory

- If features have different scales, gradient descent is steeper in some dimensions
- How can we correct for this?



Adagrad

- Adagrad update rule
  - $g \leftarrow \frac{1}{m} \nabla_\theta \sum_{i=1}^{m} L(f(x_i; \theta), y_i)$
  - $r \leftarrow r + g \cdot g$
  - $\Delta\theta \leftarrow \frac{\epsilon}{\delta + \sqrt{r}} \cdot g$, where $\delta \approx 10^{-7}$, for numerical stability
  - $\theta \leftarrow \theta + \Delta\theta$
- Shrink learning rate in each dimention according to entire history of squared gradient



Adaptive learning rates

RMSProp

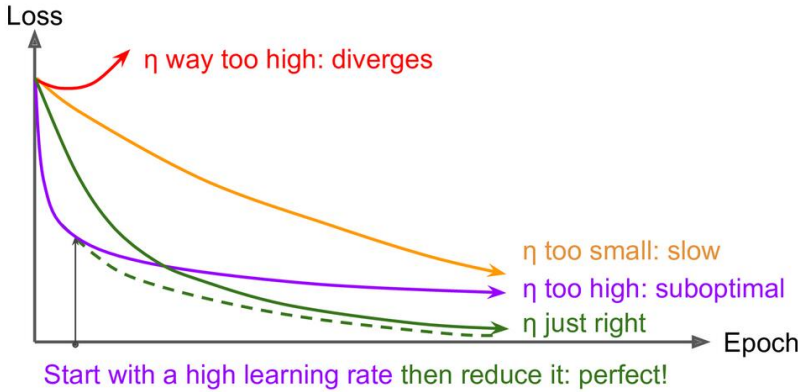
- Using entire history shrinks learning rate too much
- Exponentially decaying average, discard extreme past
- Update rule
  - $g \leftarrow \frac{1}{m} \nabla_\theta \sum_{i=1}^{m} L(f(x_i; \theta), y_i)$
  - $r \leftarrow \rho r + (1 - \rho)(g \cdot g)$, where ρ is decay rate
  - $\Delta\theta \leftarrow \frac{\epsilon}{\sqrt{\delta + r}} \cdot g$, where $\delta \approx 10^{-6}$
  - $\theta \leftarrow \theta + \Delta\theta$
- New hyperparameter ρ

Adam

- Adaptive moments — combines RMSProp and moments
- Update rule
  - Step size ϵ; two decay rates ρ_1, ρ_2; two moments, $s = r = 0$; time step $t = 0$
  - $g \leftarrow \frac{1}{m} \nabla_\theta \sum_{i=1}^{m} L(f(x_i; \theta), y_i)$
  - $s \leftarrow \rho_1 s + (1 - \rho_1) g$; $r \leftarrow \rho_2 r + (1 - \rho_2)(g \cdot g)$
  - Correct bias in first and second moments: $\hat{s} \leftarrow \frac{s}{1 - \rho_1^t}$, $\hat{r} \leftarrow \frac{r}{1 - \rho_2^t}$
  - $\Delta\theta = -\epsilon \frac{\hat{s}}{\sqrt{\hat{r}} + \delta}$; $\theta \leftarrow \theta + \Delta\theta$
- No clear theoretical justification for combinining momentum and scaling
- Fairly robust with respect to values of hyperparameters

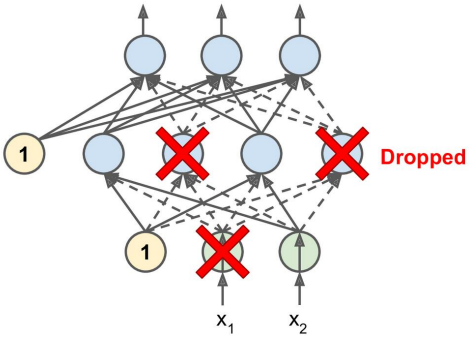
Adaptive learning rates

- Choosing a fixed learning rate is hard
- Make a learning rate a function of iteration number
- Power scheduling, exponential scheduling, piecewise constant scheduling



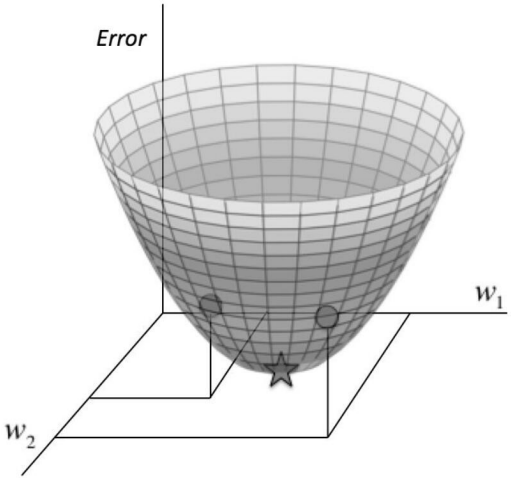
Regularization

- ℓ_1 and ℓ_2 regularization, as usual
- Dropout
  - Disable nodes with probability p
  - Analogy — multifunctional employees
  - Scale weights after training



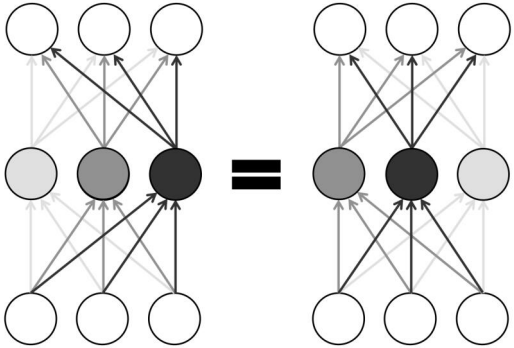
Local minima

- The loss function for regression is convex
- Gradient descent converges to global optimum
- Loss function for neural networks is not convex
- In general, gradient descent only finds local minima
- How many local minima are there?
- How does it affect gradient descent?



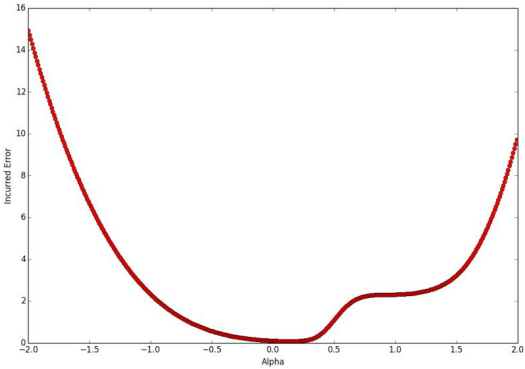
Model identifiability

- Is the model that fits the data unique?
- Non-identifiable — two or more settings of the parameters are observationally equivalent
- Symmetry
  - Fully connected network, permutations of a layer are indistinguishable
- Piecewise linear activation — ReLU
  - Scale inputs by k, multiply output by $1/k$
- Large numbers of local minima!



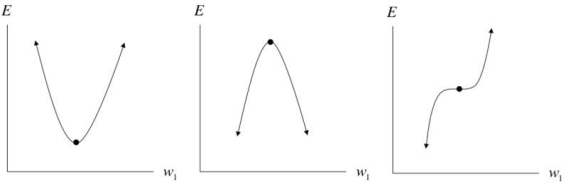
How problematic are local minima?

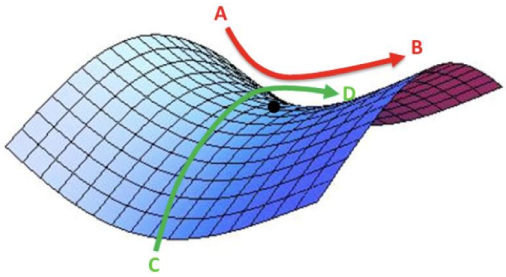
- How to measure the impact of local minima?
- Training process will see local ups and downs, but "bumpy" surface may not give a good picture
- Instead [Goodfellow et al]
  - Random initialization θ_i
  - SGD finds an optimum value θ_f
  - Check loss along the linearly interpolation $\theta_\alpha = \alpha \cdot \theta_f + (1 - \alpha) \cdot \theta_i$
  - Are there problematic local minima along the path?
- Typically, no!



Saddle points

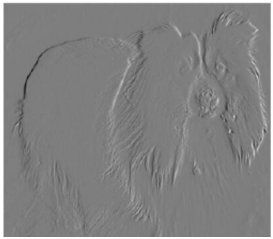
- Critical points — zero gradient
  - Minimum, maximum or inflection point
  - k critical points $\rightarrow k/3$ are minima
- In d dimensions
  - Should be minimum in d directions
  - k critical points $\rightarrow k/3^d$ are minima
- Large fraction of critical values are saddle points
- Does not seem to be a problem for SGD
- Solving directly for zero gradient is problematic



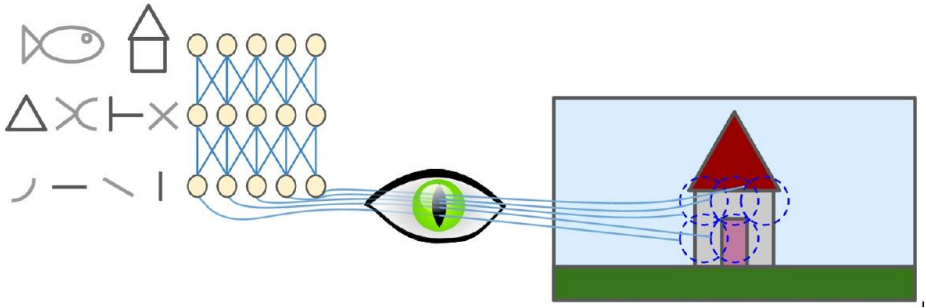


How the brain recognizes images

- Visual cortex processes images
- Experiments on cats and monkeys [Hubel, Wiesel 1959], Nobel Prize 1981
- Visual cortex organized in layers
  - Each layer detects features
  - Initial layers detect simple features — edges
  - Later layers combine features of earlier layers — detect contours, shapes, entire object
- Convolutional neural network (CNN) — layered network



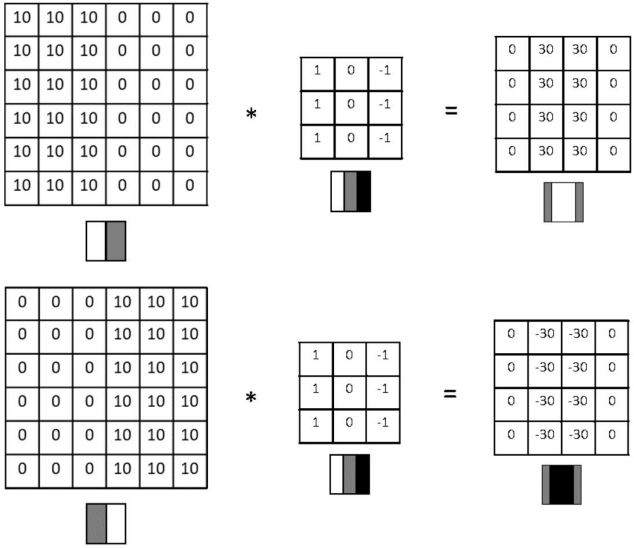
Receptive field



- Each neuron focuses on a small region — receptive field
- Vanilla neural network reads entire image as input
  - MNIST — 28×28 pixels
- Colour image, 200×200
  - Three colours — $200 \times 200 \times 3$ inputs
  - Each neuron in first layer has $120,000$ input weights
  - Multiple such neurons
- Parameter blowup, overfitting

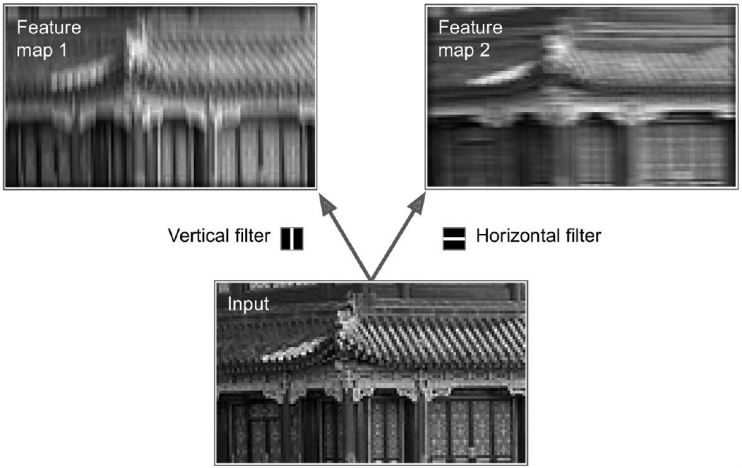
Filters and convolution

- Aggregate values over a region
  - Smoothening — take average
  - Vertical lines — difference between adjacent columns
  - Horizontal lines — difference between adjacent rows
- Pass a filter f over the image
  - Convolution — $I * f$
  - Sometimes, filter is called a convolution kernel — $I * K$
- Light to dark vertical edges
- Dark to light vertical edges



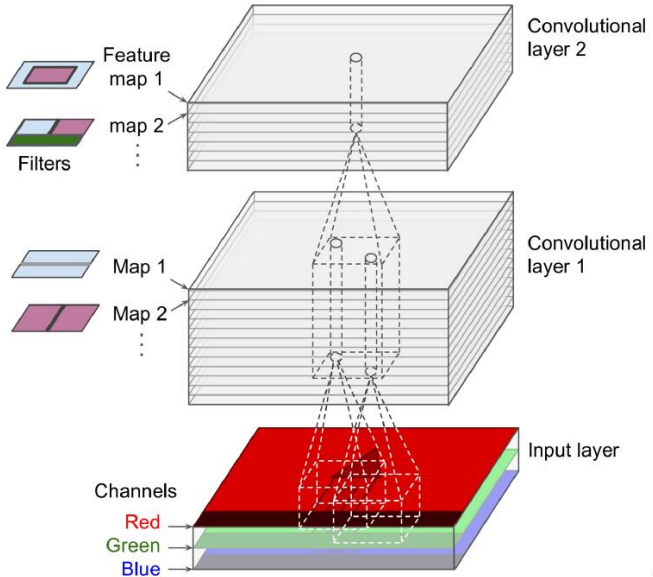
Feature maps

- Filters produce feature maps



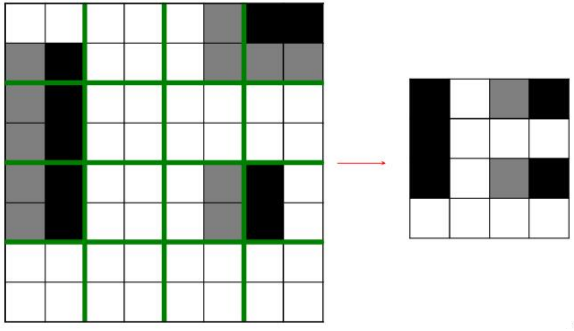
Feature maps

- Filters produce feature maps
- Colour images are split by channel
- Each layer has many feature maps
  - Array of filters, each connected to a different region
- Higher layers combine features discovered by lower layers



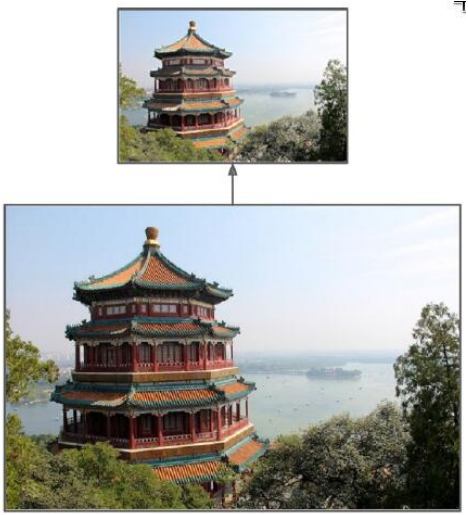
Pooling

- Filters process overlapping regions
- Pooling processes partitions
  - Subsampling, reduce dimensionality
- Most common is max-pool over 2×2 window



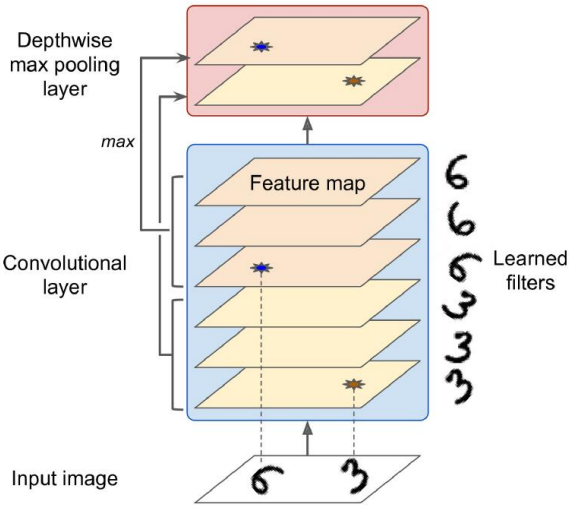
Pooling

- Filters process overlapping regions
- Pooling processes partitions
  - Subsampling, reduce dimensionality
- Most common is max-pool over 2×2 window
- Here, max-pooling reduces an image to half its size



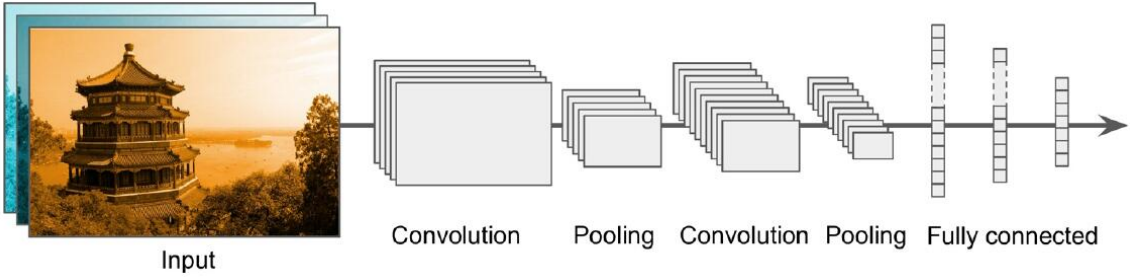
Pooling

- Filters process overlapping regions
- Pooling processes partitions
  - Subsampling, reduce dimensionality
- Most common is max-pool over 2×2 window
- Here, max-pooling reduces an image to half its size
- Can also pool depthwise — for instance, to learn features invariant to rotation



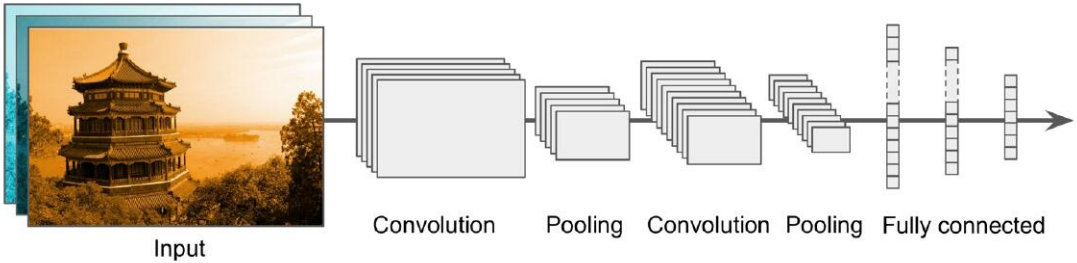
Typical CNN Architecture

- A typical CNN has multiple iterations of convolution followed by pooling
- After final pooling, conventional completely connected network



Parameter sharing

- A filter is a layer of identical nodes operating on different regions (receptive fields)
- All these nodes should behave the same
- While training, their weights are tied to each other — parameter sharing
- Thus, backward pass of backpropagation calculation is reduced
- Forward pass needs to compute individual outputs — still expensive



CNNs through the ages

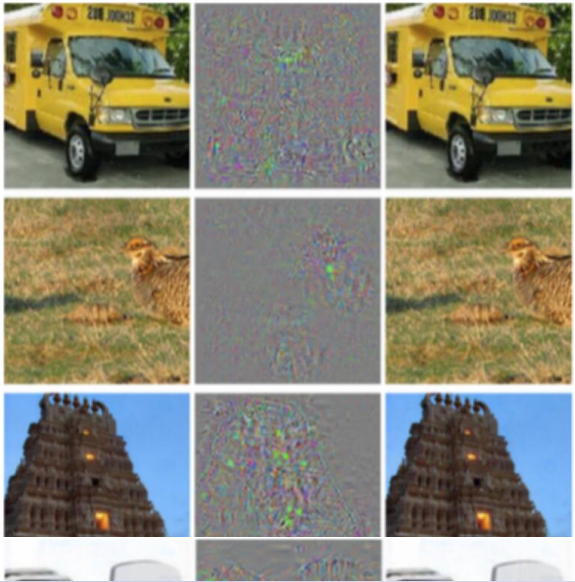
- LeNet-5 — Yann LeCun, 1998
  - Handwritten digits, MNIST data set from US postal service
- AlexNet — Alex Krizhevsky, Ilya Sutskever, Geoffrey Hinton, 2012
  - ImageNet, 14 million images, 20,000 categories, hand annotated
  - Top-five error rate — at least one of top 5 prediced labels is correct
  - 2012 ImageNet challenge, AlexNet reduced top-five error rate from 26% to 17%
  - First to add multiple convolution layers between pooling layers
  - Also some normalization layers
- GoogLeNet — Christian Szegedy et al, 2014
  - 2014 ImageNet challenge, reduced top-five error rate to 7%
  - Inception layer with 1×1 filters, operates in depth dimension, cross-channel features

CNNs through the ages

- ResNet — Kaiming He et al, 2015
  - 2014 ImageNet challenge, reduced top-five error rate to under 3.6%
  - 152 layers!
  - Skip connections to speed up learning
    - Input to a layer is added to output of a higher layer
    - Higher layer learns $h(x) - x$ rather than $h(x)$ — residual learning
    - Accelerates learning through multiple layers
- Xception, Chollet, 2016
- SENet, Hu et al, 2017

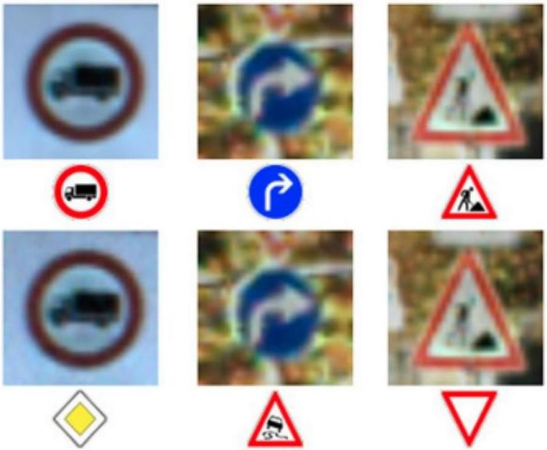
Understanding how ML models work

- Imperceptible changes alter output
  - All images in the last column are classified as ostrich!
- Random images are given meaning



Adversarial attacks

- Self driving cars and traffic signs

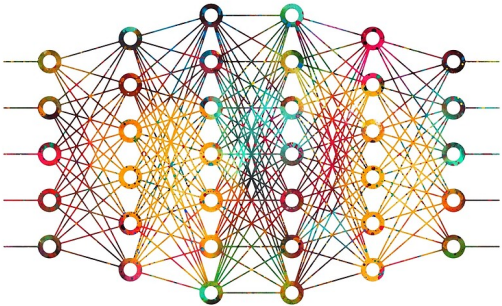


Adversarial attacks

- Self driving cars and traffic signs
- Can be achieved with "stickers"

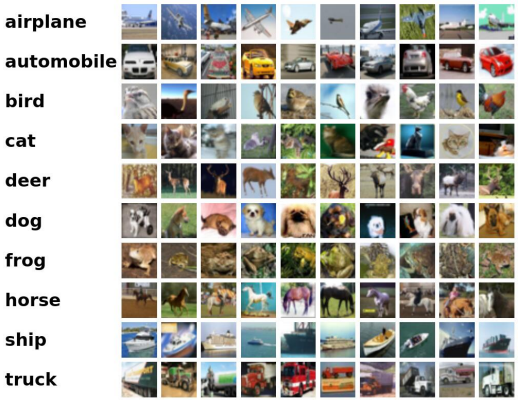
Explainability vs interpretability

- Can we explain the behaviour of a complex model?
- Build interpretable models to start with



Brittleness

- CIFAR-10 dataset, 10 classes, 50,000 training images



Brittleness

- CIFAR-10 dataset, 10 classes, 50,000 training images
- Yacht image classified correctly 84% of the time, across different training runs — robust

Brittleness

- CIFAR-10 dataset, 10 classes, 50,000 training images
- Yacht image classified correctly 84% of the time, across different training runs — robust
- Removing just the 9 images below from training data reduces accuracy on this image to 34%
- Removing 11 more reduces accuracy to 10% — as bad as random guessing!