Lecture 4: Training Deep Neural Networks

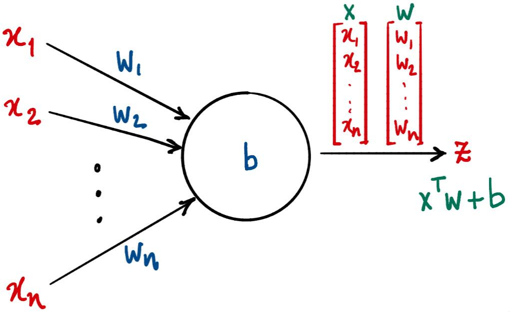
Madhavan Mukund

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

Advanced Machine Learning
September–December 2021

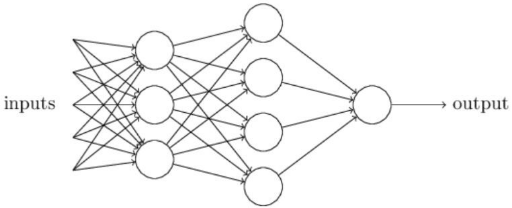
Linear separators and perceptrons

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



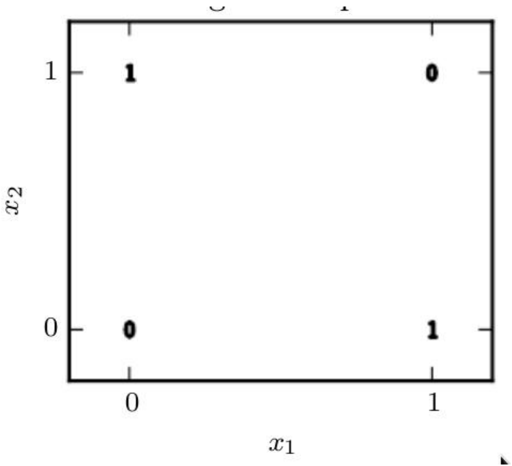
Linear separators and perceptrons

- Perceptrons define linear separators
  $x^T w + b$
  - $x^T w + b > 0$, classify Yes $(+1)$
  - $x^T w + b < 0$, classify No (-1)
- Network of perceptrons still defines only a linear separator



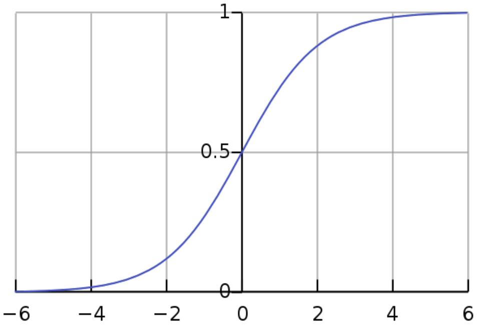
Linear separators and perceptrons

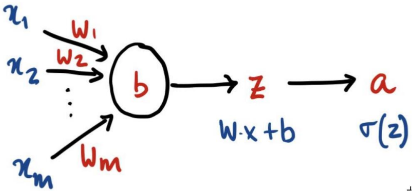
- Perceptrons define linear separators
  $x^T w + b$
  - $x^T w + b > 0$, classify Yes $(+1)$
  - $x^T w + b < 0$, classify No (-1)
- Network of perceptrons still defines only a linear separator
- Linear separators cannot describe XOR



Linear separators and perceptrons

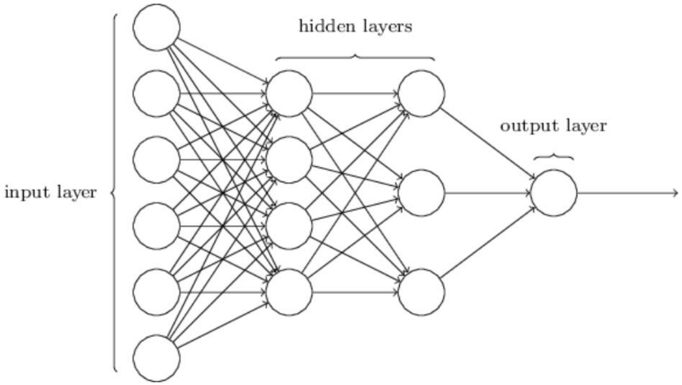
- Perceptrons define linear separators
  $x^T w + b$
  - $x^T w + b > 0$, classify Yes $(+1)$
  - $x^T w + b < 0$, classify No (-1)
- Network of perceptrons still defines only a linear separator
- Linear separators cannot describe XOR
- Introduce a non-linear activation function
  - Traditionally sigmoid,
    $\sigma(z) = 1/(1 + e^{-z})$





(Feed forward) Neural networks

- Acyclic network of perceptrons with non-linear activation functions
- Ingredients
  - Output layer activation function
  - Loss function for gradient descent
  - Hidden layer activation functions
  - Network architecture: Interconnection of layers
  - Initial values of weights and biases



Training a neural network

- Backpropagation — efficient implementation of gradient descent for neural networks
- Forward pass, compute outputs, activation values
- Backward pass, use chain rule to compute all gradients in one scan
- Stochastic gradient descent (SGD)
  - Update parameters in minibatches
  - Epoch: set of minibatches that covers entire training data
- Difficulties: slow convergence, vanishing and exploding gradients

$\xrightarrow{\text{Compute } z, a}$

$\xleftarrow{\hspace{3em}}$

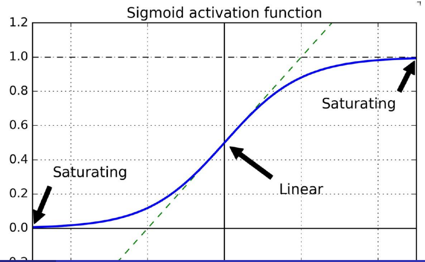
Compute $\frac{\partial C}{\partial z^l_k}, \frac{\partial C}{\partial w^l_{kj}}, \frac{\partial C}{\partial b^l_k}$

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

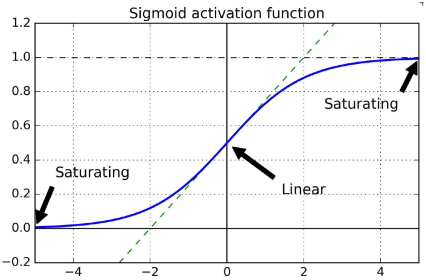
Compute z, a $\longrightarrow$

$\longleftarrow$ Compute $\frac{\partial C}{\partial z^l_k}, \frac{\partial C}{\partial w^l_{kj}}, \frac{\partial C}{\partial b^l_k}$



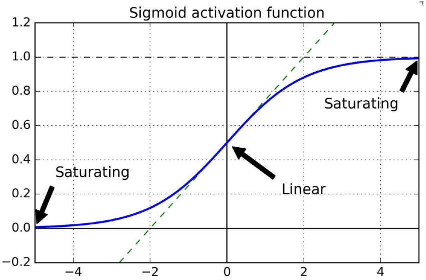
Initializing neural networks

- 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, $\mathcal{U}(-r, r)$, $r = \sqrt{\dfrac{3}{fan_{avg}}}$



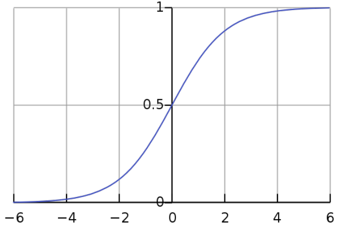
Initializing neural networks

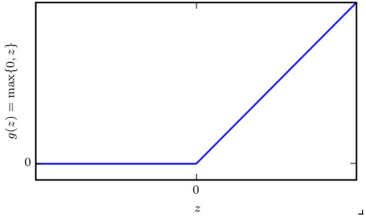
- Let $fan_{avg} = (fan_{in} + fan_{out})/2$
- Initialize with
  - Gaussian, $\mathcal{N}(0, 1/fan_{avg})$
  - Uniform, $\mathcal{U}(-r, r)$, $r = \sqrt{\dfrac{3}{fan_{avg}}}$
- [Yann LeCun, 1990s] earlier proposed the same with fan_{avg} replaced by fan_{in}
  - Equivalent if $fan_{in} = fan_{out}$
- Other choices for specific activation function
  - ReLU, [He et al, 2015], $\mathcal{N}(0, 2/fan_{in})$



Non-saturating activation functions

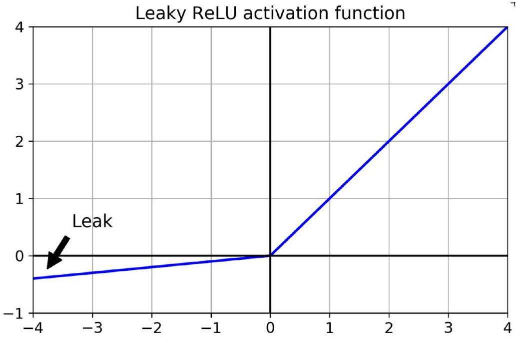
- Sigmoid was initially chosen as a "smooth" step
- Rectified linear unit (ReLU): $g(z) = \max(0, z)$
  - Fast to compute
  - Non-differentiable point not a bottleneck
- "Dying ReLU"
  - Neuron dies — weighted sum of outputs is negative for all training samples
  - With a large learning rate, half the 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
- PReLU — parametric ReLU [He et al, 2015]
  - α is learned during training
  - Often outperforms ReLU, but could lead to overfitting

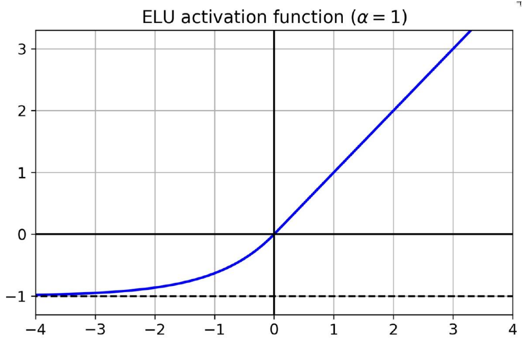


Non-saturating activation functions

- ELU — Exponential Linear Unit [Clevert et al, 2015]

$$ELU_\alpha(z) = \begin{cases} \alpha(e^z - 1) & \text{if } z < 0 \\ z & \text{if } z \geq 0 \end{cases}$$

  - Training converges faster
  - Computing exponential is slower
  - In practice, slower than ReLU

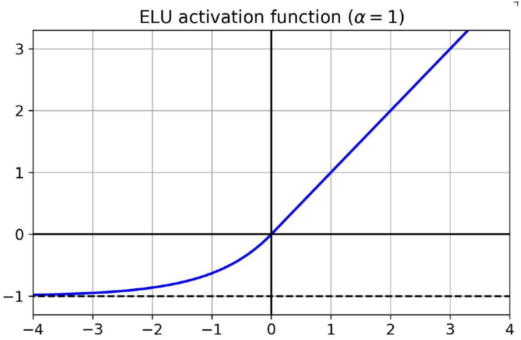


Non-saturating activation functions

- SELU — Scaled ELU
  [Klambauer et al, 2017]

$$SELU_\alpha(z) = \lambda \begin{cases} \alpha(e^z - 1) & \text{if } z < 0 \\ z & \text{if } z \geq 0 \end{cases}$$

  - Self-normalizing — output of each layer preserves mean 0 and standard deviation 1 during training
  - Use LeCun initialization, $\mathcal{N}(0, 1/fan_{in})$

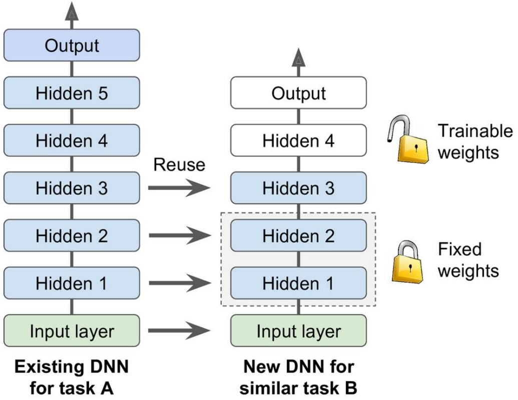


Batch normalization [Joffe, Szegedy 2015]

- Good activation function and initialization mitigates vanishing/exploding gradients
- May still recur during training
- Add batch normalization (BN) layers
  - Estimate mean μ_B and variance σ_B^2 for inputs across minibatch
  - Zero-centre and normalize each input
    $\hat{x}_i = \dfrac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$
  - Scale and shift $z_i = \lambda \cdot \hat{x}_i + \beta$
  - Learn optimal scaling and shifting parameters for each layer
- At input, BN layer avoids need for standardizing
- Difficulties
  - Mean and variance differ across minibatches
  - How to estimate parameters for entire dataset?
  - Practical solution: maintain a moving average of means and standard deviations for each layer
- Batch normalization greatly speeds up learning rate
- Even works 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



Still to come

- Optimizing rate of updates in backpropagation
- How problematic are local minima?
- Identifying and dealing with unstable gradients