Introduction to Artificial Intelligence

Lecture 4: Neural networks I

September 15, 2025

Recap: Linear regression with groups

- Suppose we have G groups, $1,2,\ldots,G$, in total
  - Example: {water bird, water background}, {water bird, land background}, {land bird, water background}, {land bird, land background}
- To encode this prior into the model, introduce a per-group loss

$$\hat{L}_g(\beta) = \frac{1}{n_g} \sum_{(x,y)\in D_g} \ell(x, y; \beta)$$

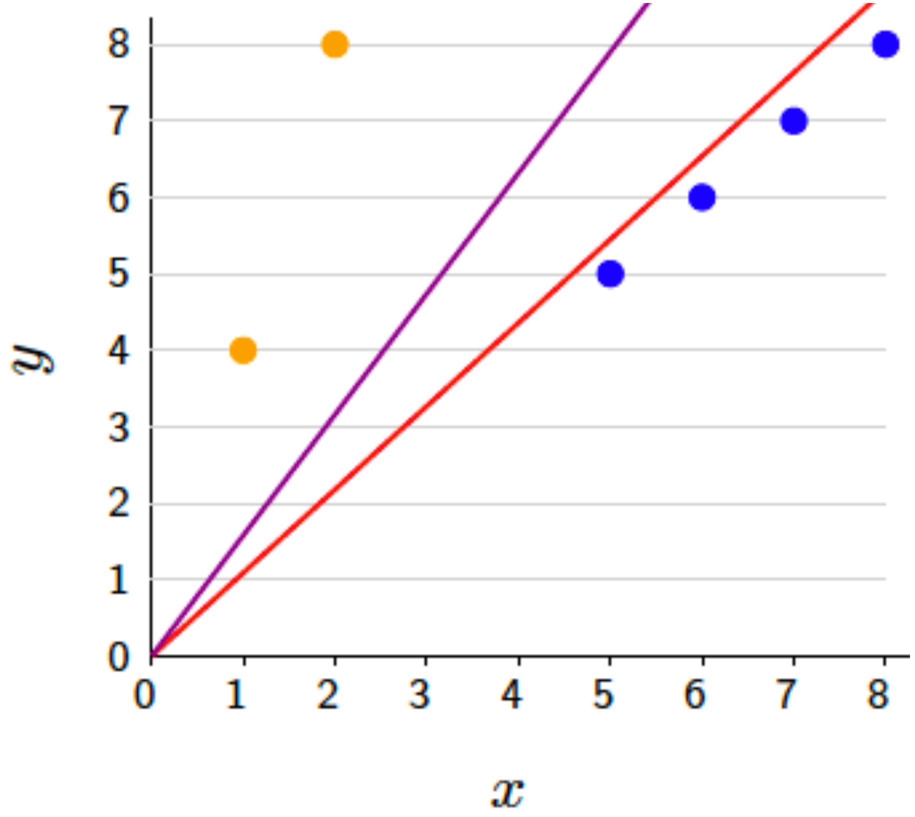
- Instead of minimizing the average loss, minimize the maximum group loss instead

$$\min_{\beta} \max_{g} \hat{L}_g(\beta)$$

Recap: Average loss vs. maximum group loss

- Illustration in a toy example with six data points, separated into two groups

x	y	g
1	4	A
2	8	A
5	5	B
6	6	B
7	7	B
8	8	B



Average loss vs. maximum group loss

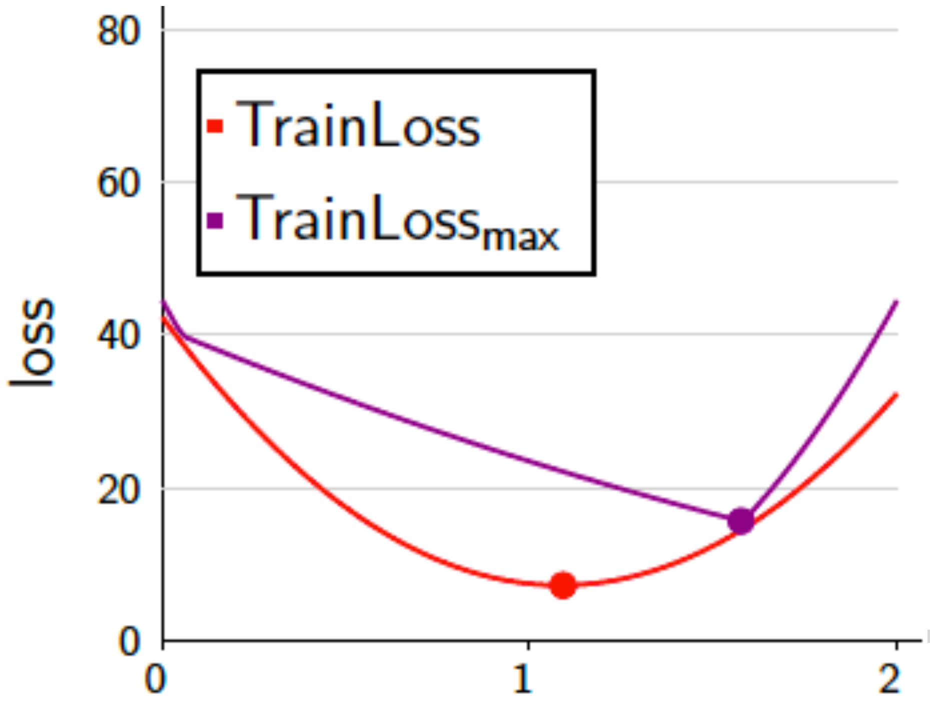
- TrainLoss:

$$\frac{1}{6}\cdot\left((w-4)^2+(2w-8)^2+(5w-5)^2+(6w-6)^2+(7w-7)^2+(8w-8)^2\right)=\frac{5}{6}(w-4)^2+29(w-1)^2$$

- TrainLossMax:

$$\max\left(\frac{1}{2}\left((w-4)^2+(2w-8)^2\right),\frac{1}{4}\left((5w-5)^2+(6w-6)^2+(7w-7)^2+(8w-8)^2\right)\right)=$$
$$\max\left(\frac{5}{2}(w-4)^2,43\frac{1}{2}(w-1)^2\right)$$

x	y	g
1	4	A
2	8	A
5	5	B
6	6	B
7	7	B
8	8	B



Recap: Gradient descent

- The algorithm is as follows:

$w \leftarrow$ any point in the parameter space

While not converged

For each $w_i \in w$

$$w_i \leftarrow w_i - \alpha \cdot \frac{\partial Loss(w)}{\partial w_i}$$

- α: step size, or learning rate

Chain rule

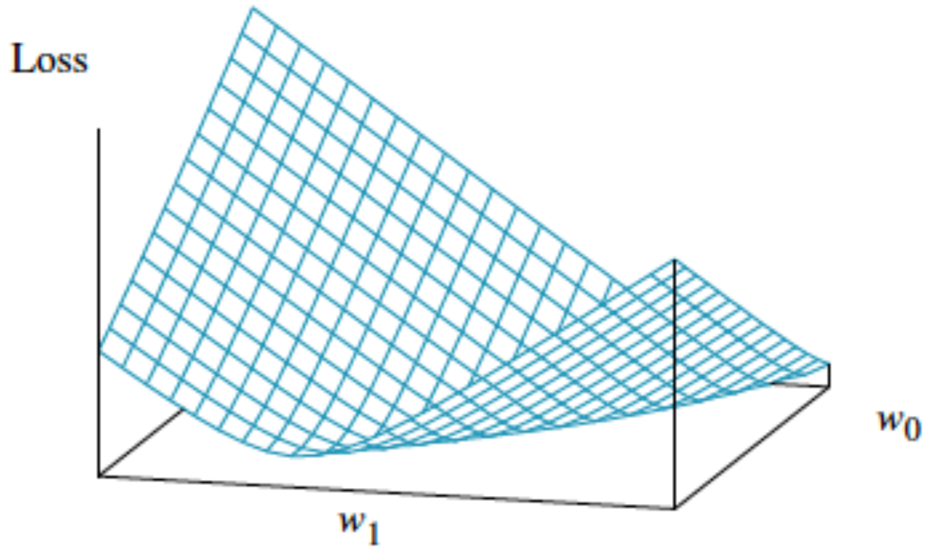
- Chain rule of calculus: $\frac{\partial g(f(x))}{\partial x} = g'(f(x))\frac{\partial f(x)}{\partial x}$
- In the case of squared loss:

$$\frac{\partial Loss(w)}{\partial w_i} = \frac{\partial (y - f_w(x))^2}{\partial w_i} = 2(y - f_w(x))\frac{\partial (y - f_w(x))}{\partial w_i} = 2(y - f_w(x))\frac{\partial (y - (w_1 x + w_0))}{\partial w_i}$$

- Exercise: Applying this to both w_0 and w_1?
- Next: Apply this update to the gradient descent algorithm.

Stochastic gradient descent

- Randomly selects a small number of training examples at each step



Lecture plan

- Neural networks and deep learning
  - **Feedforward neural networks**

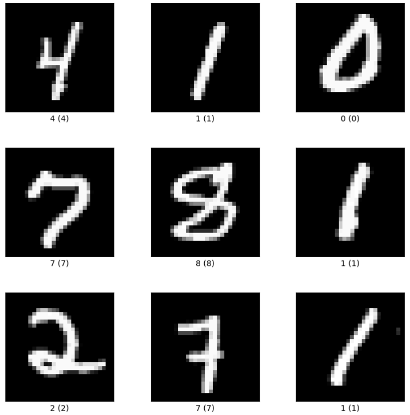
Low level **Machine learning** **High level**

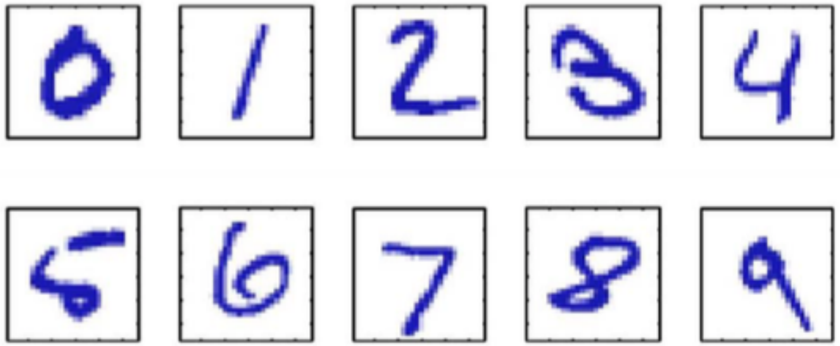
Supervised learning

Neural networks and deep learning

Handwritten digit recognition

- Input: handwritten digits from 0 to 9 in black and white

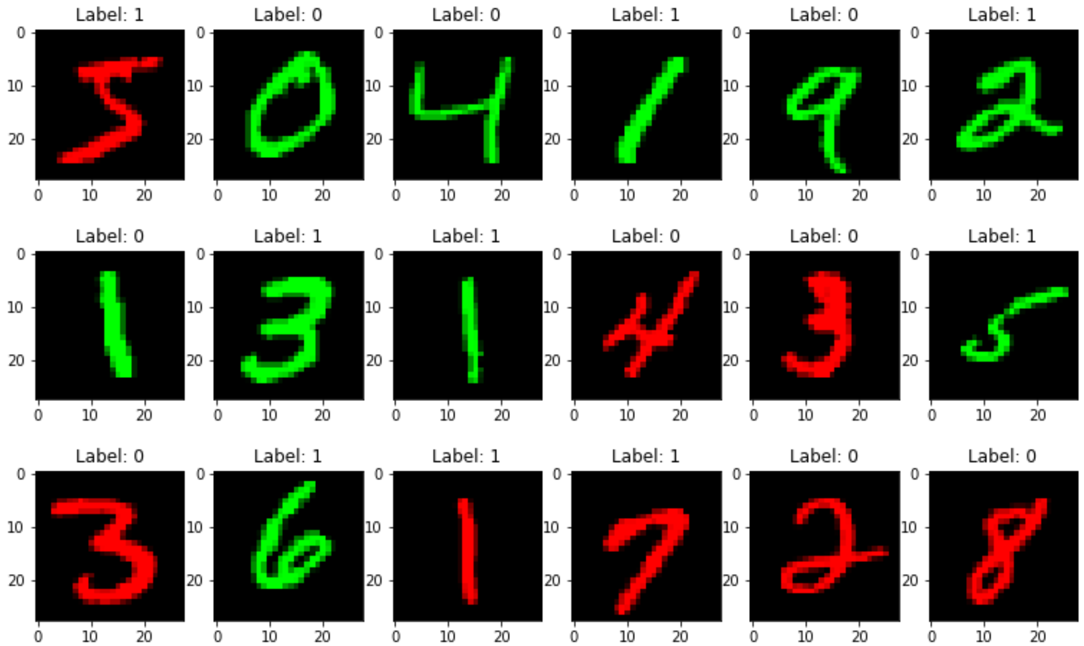


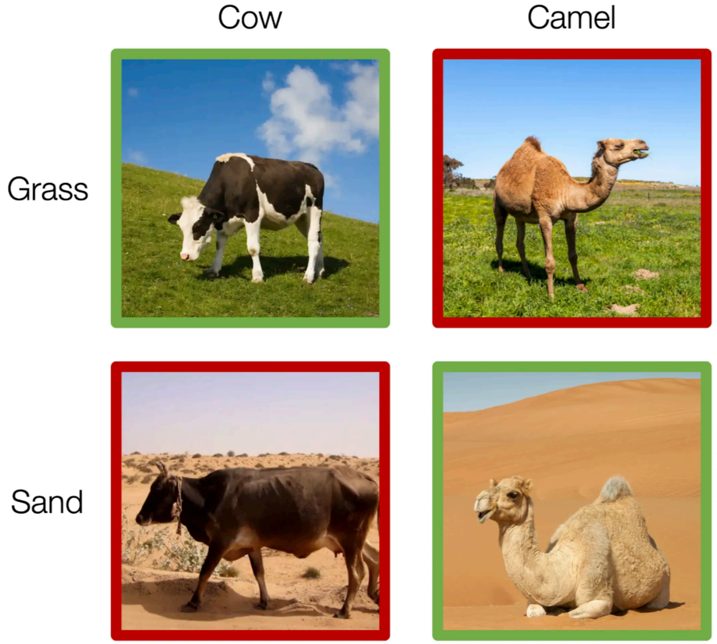


- **MNIST:** 50,000 handwritten digits for training; 5,000 for validation; 5,000 for testing

Colored digits

- Colored MNIST: Colored digits in a black ground
  - Input is represented from 3 times 28 times 28 pixels

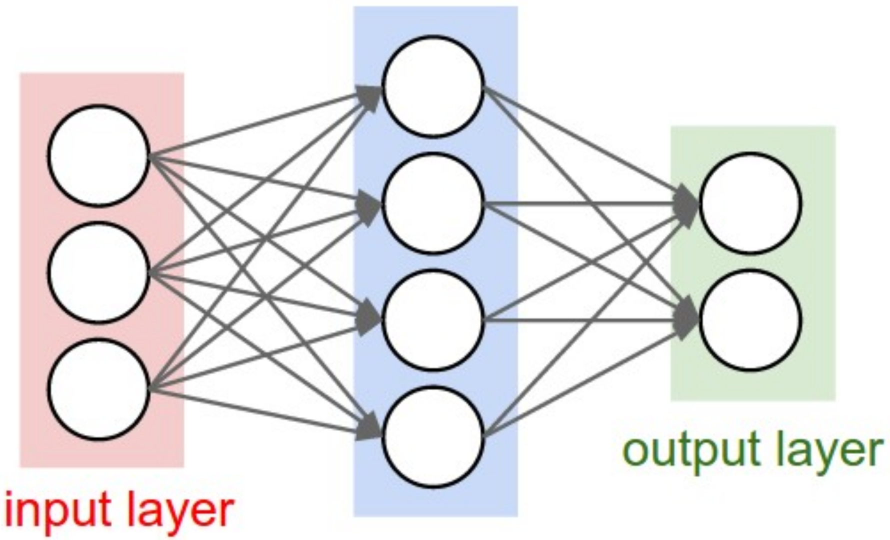




  - A naive model may simply predict the digit based on its color---a problem known as **spurious correlation**
  - Link: https://github.com/facebookresearch/InvariantRiskMinimization/blob/main/code/colored_mnist/main.py

Feedforward neural networks

- Example of a feedforward neural network



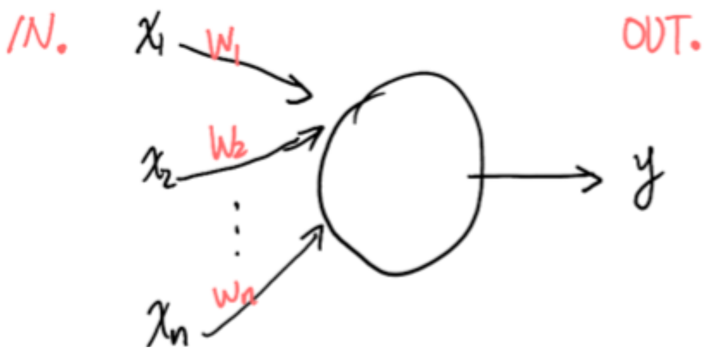
- This simple network works well on MNIST and other handwritten digit recognition examples

Lecture plan

- **Deep:** circuits are typically organized into many **layers**
- **Deep learning:** most widely used approach for applications such as visual object recognition, machine translation, speech recognition, and robotics
- **Today: feedforward neural networks**
  - **Artificial neuron: Perceptron (https://en.wikipedia.org/wiki/Perceptron)**
  - Networks as a complex function
  - Neural network architecture
  - Learning algorithms

An artificial neuron

- Perceptron is a type of artificial neuron



- Input: n real values $x_1, x_2, \ldots, x_n$
- Weight parameters $w_1, w_2, \ldots, w_n$ connecting every input to the neuron
- Output
  - $y = 0$, if $\sum_{j=1}^{n} w_j x_j + b < 0$
  - $y = 1$, if $\sum_{j=1}^{n} w_j x_j + b \geq 0$

Example

- There is a new restaurant opened near Northeastern
  - x_1 = "Is the dinner over \$30 per person?"; $w_1 = -30$
  - x_2 = "Is the parking fee over \$10?"; $w_2 = -10$
  - x_3 = "Is the wait time over half an hour?"; $w_3 = -10$

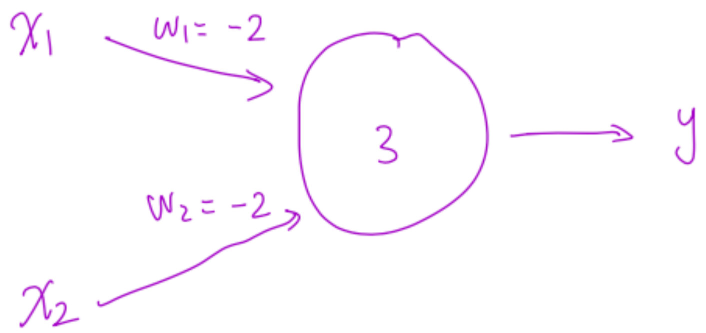
- Budget $\mathrm{b} = 40$
  - If $x_1 = 1, x_2 = 1, x_3 = 0$, then $y = 1$
  - If $x_1 = 0, x_2 = 1, x_3 = 1$, then $y = 1$
  - If $x_1 = 1, x_2 = 1, x_3 = 1$, then $y = 0$

Vector notation

- Vector notation allows us to write the operation within an artificial neuron more concisely
  - $\langle w, x \rangle + b \geq 0 \Rightarrow y = 1$
  - $\langle w, x \rangle + b < 0 \Rightarrow y = 0$
  - $w = [w_1, w_2, \ldots, w_n]$ including all weight parameters in a vector
  - $b =$ bias: measures how easy it is to activate the neuron

Example

- Compute elementary logical functions
- **Example**: Use a perceptron to represent Negated AND



- If $x_1 = 1, x_2 = 1$, then $y = 0$
- If $x_1 = 1, x_2 = 0$, then $y = 1$
- If $x_1 = 0, x_2 = 1$, then $y = 1$
- If $x_1 = 0, x_2 = 0$, then $y = 1$

Sigmoid neuron

- Perceptron is susceptible to small perturbations
  - If $\langle w, x \rangle + b \approx \epsilon$, then a small change in x flips y: suppose $\epsilon = 0.01$, but the perturbation reduces ϵ by 0.02; this flips y from 1 to 0
- Sigmoid neurons do not suffer from this problem: It provides a nice approximation of the zero-one function

$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$

- $z = \langle w, x \rangle + b$
  - If $z \geq 0$, then $\mathrm{y} = \sigma(z) \geq 0.5$
  - If $z < 0$, then $\mathrm{y} = \sigma(z) < 0.5$

Sigmoid neuron

- Intuition
  - When $z = \langle w, x \rangle + b$ is very large (say ≥ 10), y is very close to one
  - When $z = \langle w, x \rangle + b$ is very small (say < -10), y is very close to zero
- One can change the slope of sigmoid neurons by inserting a temperature parameter t

$$\sigma(z) = \frac{1}{1 + \exp(-t \cdot z)}$$

- Sigmoid neurons are differentiable: can run auto-differentiation in PyTorch or TensorFlow

Mid-class break

- Link to the survey:

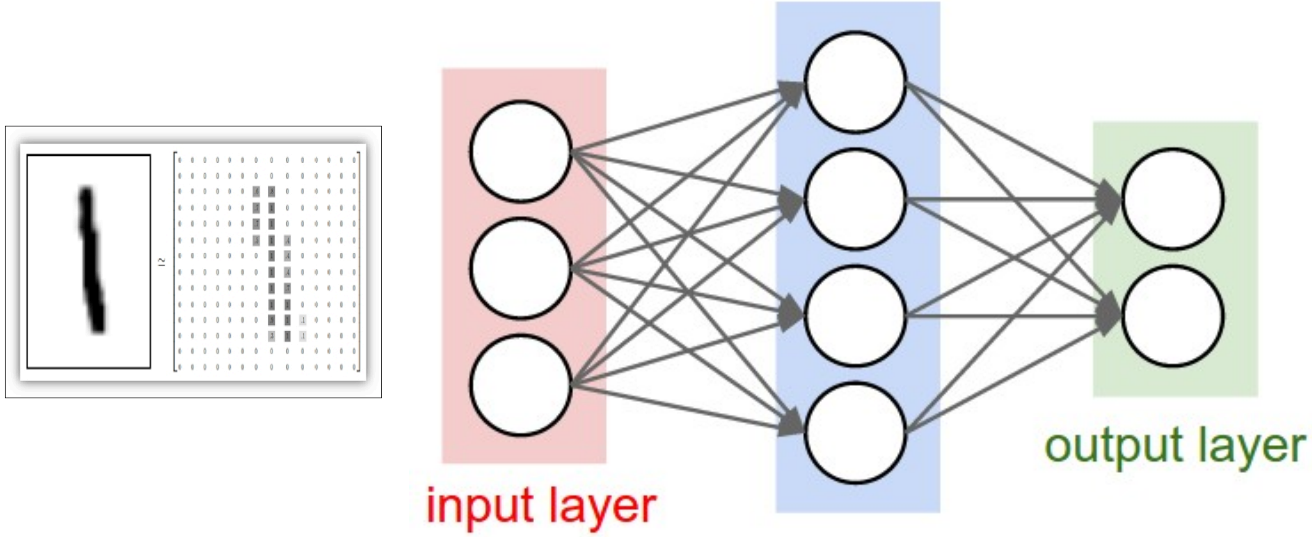
https://forms.gle/MdPYJV72AT7tjPSG6

Lecture plan

- **Feedforward neural networks**
  - Artificial neuron: Perceptron
  - **Networks as a complex function**
  - Neural network architecture
  - Learning algorithms

Input-output behavior

$\boldsymbol{\sigma(z)}$ is the neuron's activation function

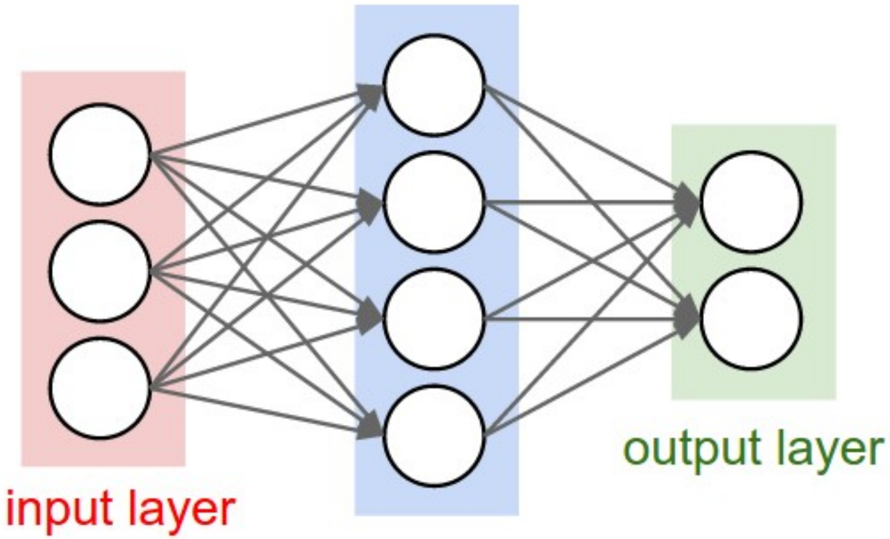


hidden layer with four neurons

Prediction over {0,1,2,3,4,5,6,7,8,9}

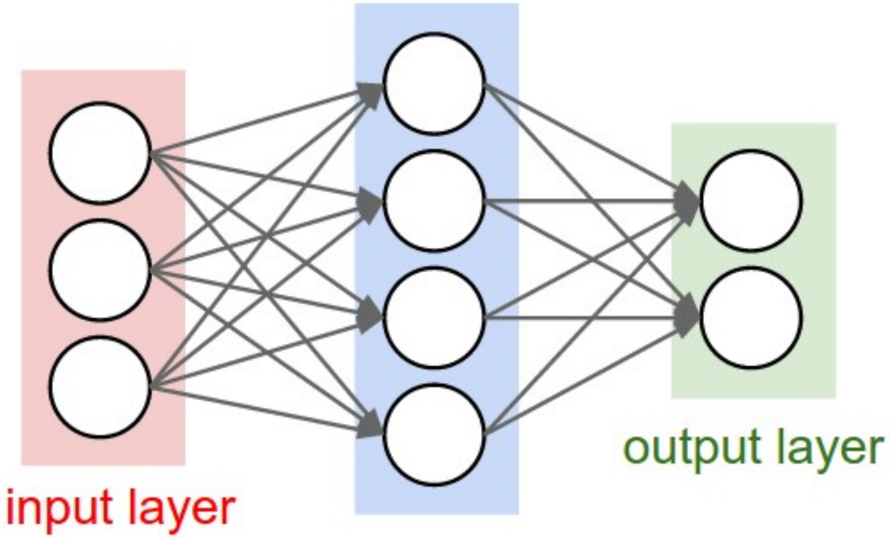
Design choices

- Width: Number of neurons in the hidden layer
- In the following example, width is four



Design choices

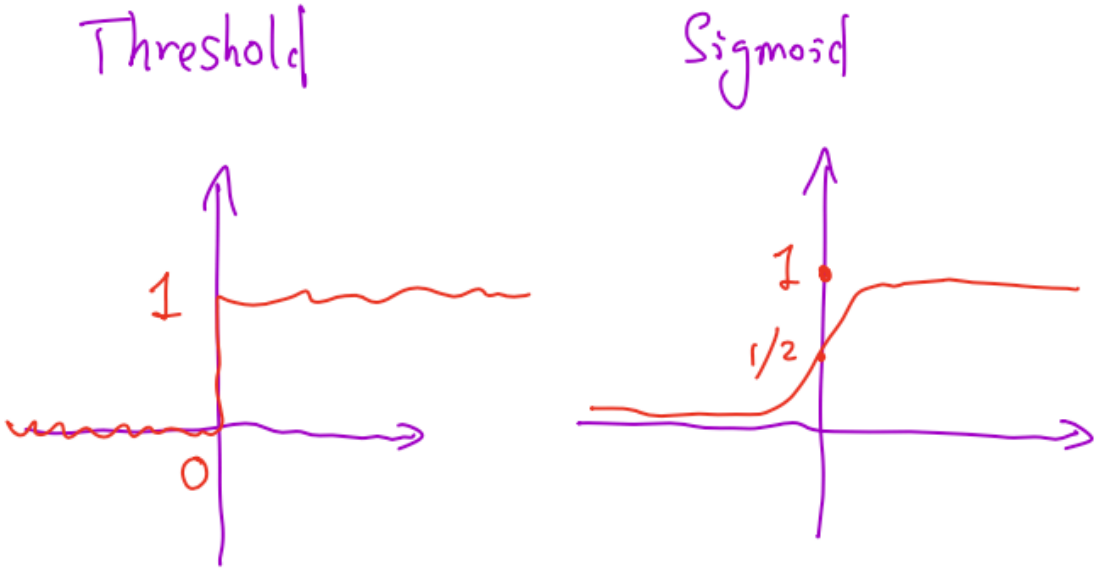
- Width also determines the number of parameters in the network



- Number of parameters: 4 times (3 + 2) plus 4 is 24
  - Width times (number of neurons in the input layer + number of neurons in the output layer) + number of hidden-layer neurons

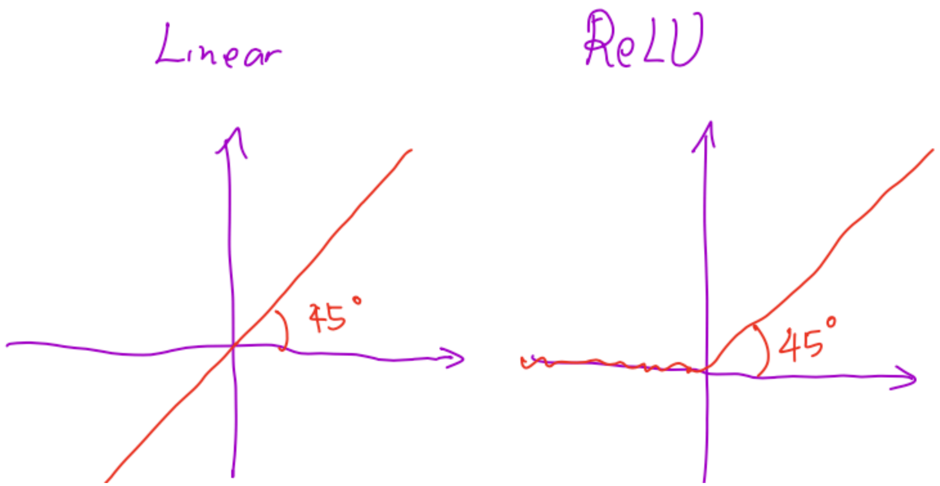
Design choices

- Activation function $\sigma: \mathbb{R} \to \mathbb{R}$
- Threshold function: $\sigma(z) = 0 \; if \; z \leq 0, 1 \; if \; z > 0$
- Sigmoid function: $\sigma(z) = \frac{1}{1+\exp(-z)}$



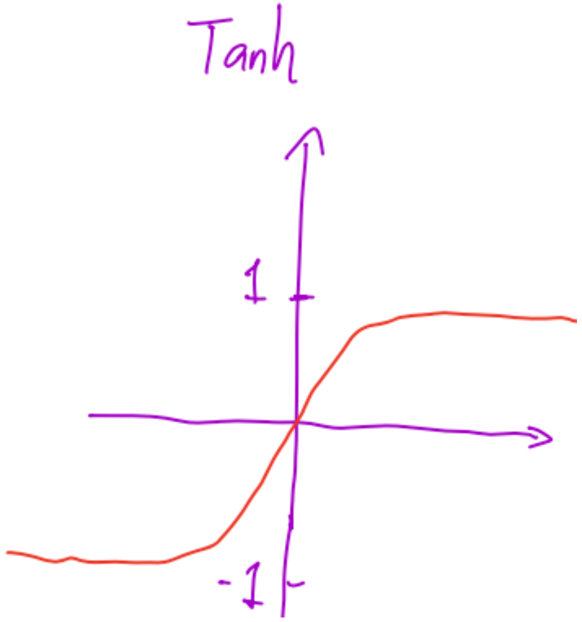
Design choices

- Activation function $\sigma: \mathbb{R} \to \mathbb{R}$
- Linear function: $\sigma(z) = z$
- Rectified linear units (ReLU): $\sigma(z) = \max(z, 0)$



Design choices

- Activation function $\sigma: \mathbb{R} \to \mathbb{R}$
- Tanh: $\sigma(z) = \frac{e^{2z}-1}{e^{2z}+1}$, similar to sigmoid but allows for the -1 mode



- Tanh is used in transformers

Summary of activation functions

- Threshold function: $\sigma(z) = 0 \; if \; z \leq 0; \sigma(z) = 1 \; if \; z > 0$
- Sigmoid function: $\sigma(z) = \frac{1}{1+\exp(-z)}$
- Linear function: $\sigma(z) = z$
- Rectified linear units (ReLU): $\sigma(z) = \max(z, 0)$
- Tanh: $\sigma(z) = \frac{e^{2z}-1}{e^{2z}+1}$, similar to sigmoid but allows for -1

Quick question

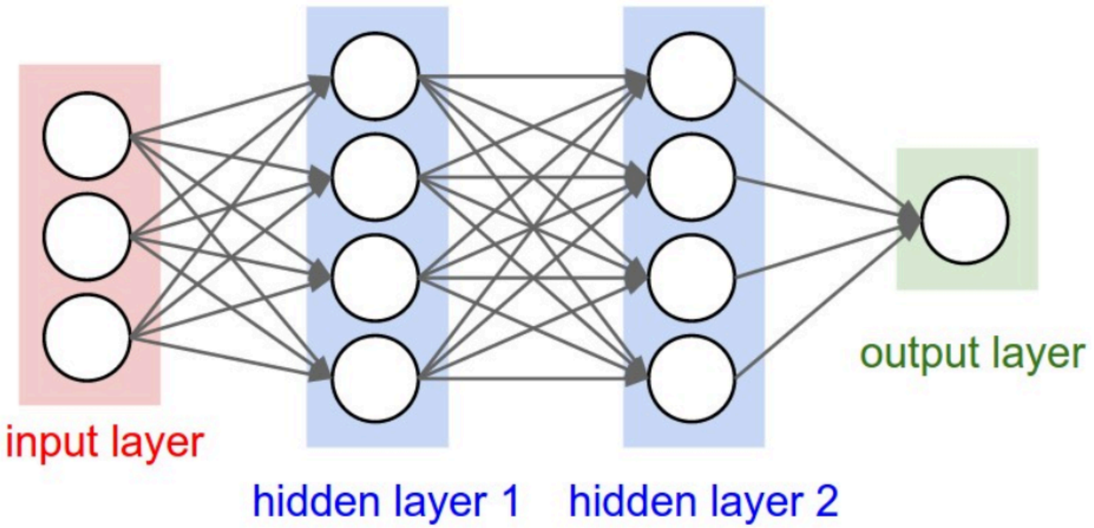
- How shall we set the number of output neurons?
  - In the MNIST example, we want to use ten output nodes; one for each class from zero to nine
  - For binary classification, the number of output nodes is two
  - For regression, the number of output nodes is one

Lecture plan

- **Feedforward neural networks**
  - Artificial neuron: Perceptron
  - Networks as a complex function
  - **Neural network architecture**
  - Learning algorithms

Multi-layer neural networks

- Extending two-layer neural network to multi-layer neural network



Other types of neural networks

- Feedforward neural networks receive the input data in no particular order
- This works well for images and other types of data that do not require sequential information
- For text data, we process the data in a sequential order: transformer and self-attention mechanisms are ideally suited for that

Lecture plan

- **Feedforward neural networks**
  - Artificial neuron: Perceptron
  - Networks as a complex function
  - Neural network architecture
  - **Learning algorithms**

Quadratic loss

- Given a prediction u for a data point x with label y

$$l(x) = (u - y)^2$$

- Suitable for regression problems with neural networks
- Apply chain rule to get the gradient $\nabla_w l(f_w(x), y)$
- Run the gradient descent algorithm

Cross-entropy loss

- Given a prediction for every label $y \in \{1,2,\dots,k\}$, let u be this vector
- Softmax maps u into a probability distribution: $\frac{\exp(u_y)}{\sum_{i=1}^{k}\exp(u_i)}$
- Apply negative log to softmax:

$$\ell(u) = -\log\frac{\exp(u_y)}{\sum_{i=1}^{k}\exp(u_i)}$$

- Example: for MNIST, the label space is $\{0,1,2,\dots,9\}$
  - Softmax output for **1**: [0.01, **0.9**, 0.01, 0.01, 0.01, 0.01, 0.01, 0.02, 0.01, 0.01]

PyTorch

- L is the label space
- y_n is the label of x_n
- $x_{n,c}$ is the softmax output probability of x_n for label c

CrossEntropyLoss = Negative Log Likelihood applied to SoftMax

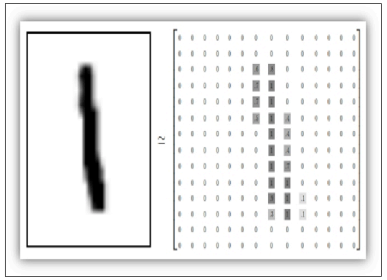
```
CLASS torch.nn.CrossEntropyLoss(weight=None, size_average=None, ignore_index=- 100,
      reduce=None, reduction='mean', label_smoothing=0.0) [SOURCE]
```

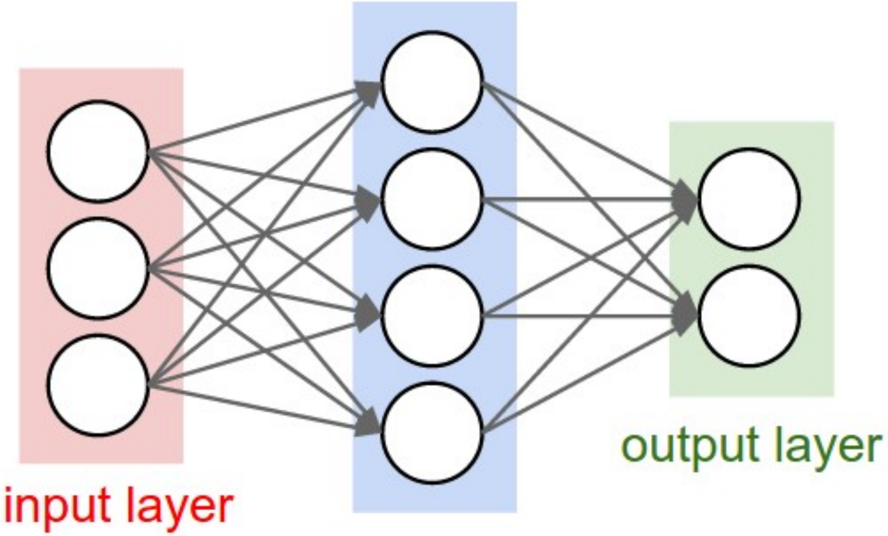
$$\ell(x, y) = L = \{l_1, \ldots, l_N\}^\top, \quad l_n = -w_{y_n} \log \frac{\exp(x_{n,y_n})}{\sum_{c=1}^{C} \exp(x_{n,c})} \cdot 1\{y_n \neq \text{ignore\_index}\}$$

https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html

Summary

- Classifying handwritten digits with two-layer neural nets
  - Input layer takes an input, often in vector or matrix format
  - Hidden layer uses an activation function (ReLU for handwritten digits)
  - Output layer applies softmax to convert the hidden-layer representation to a probability distribution
  - Use gradient descent to minimize the cross-entropy loss and train parameters





Prediction over {0,**1**,2,3,4,5,6,7,8,9}

Softmax output [0.01, **0.9**, 0.01, 0.01, 0.01, 0.01, 0.01, 0.02, 0.01, 0.01]