Supervised Machine Learning and Learning Theory

Lecture 11: Introduction to neural networks

October 11, 2024

Warm-up questions

- For bagging and random forests, how do we run cross-validation, and compute the cross-validation error?
  - A: After bootstrap there are ~37% data left, we use that as the holdout set to compute CV error
- What is the advantage of random forests compared to bagging?
  - A: RF captures more dependencies of feature subsets due to its sampling of features during construction
- What are the key design parameters in random forests? And how should we adjust them?
  - A: # trees, # features (or columns) per sample, depth; we adjust them with CV
- What about gradient boosted trees compared to random forests? Also describe their differences
  - A: Gradient boosted trees are sequential while RFs are parallel; Gradient boosted trees are deterministic, and each tree only has few splits (unlike RFs where a tree can have many splits)

Gradient boosting

X_1	Y_1
X_2	Y_2
X_3	Y_3
X_4	Y_4
X_5	Y_5

$$r_i^1 \leftarrow Y_i - \lambda \hat{f}^1(X_i)$$

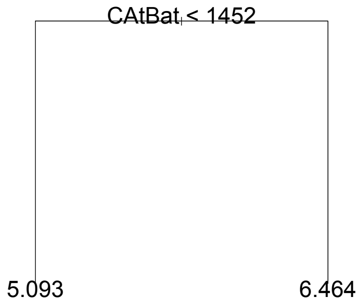
X_1	r_1^1
X_2	r_2^1
X_3	r_3^1
X_4	r_4^1
X_5	r_5^1

$$r_i^2 \leftarrow r_i^1 - \lambda \hat{f}^2(X_i)$$

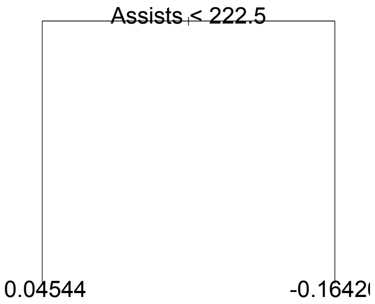
X_1	r_1^2
X_2	r_2^2
X_3	r_3^2
X_4	r_4^2
X_5	r_5^2

…

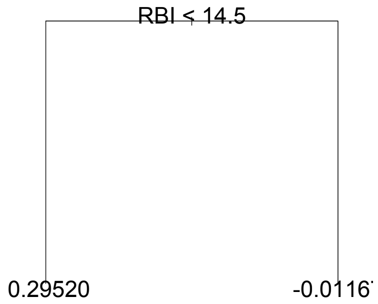
$\hat{f}^1(x)$



$\hat{f}^2(x)$



$\hat{f}^3(x)$



$$\hat{f}(x) = \lambda \hat{f}^1(x) + \lambda \hat{f}^2(x) + +\lambda \hat{f}^3(x) + \cdots + \lambda \hat{f}^B(x)$$

AdaBoost

- Another way of boosting: suppose $Y \in \{-1,1\}$

		Initial weight
X_1	Y_1	$1/5$
X_2	Y_2	$1/5$
X_3	Y_3	$1/5$
X_4	Y_4	$1/5$
X_5	Y_5	$1/5$

$$Error = \frac{1}{n}\sum_i I(\hat{f}(X_i) \neq Y_i) = \frac{2}{5}$$

$$\frac{1}{2}\log\frac{1 - Total\ Error}{Total\ Error} = \frac{1}{2}\log\frac{1-2/5}{2/5} = 0.088$$

Fitted tree $\hat{f}^1(x)$
Correctly predict all samples besides Y_3 and Y_5

Increase sample weight for the sample that was incorrectly classified
Decrease sample weight for the sample that was correctly classified

AdaBoost

		Initial weight
X_1	Y_1	$1/5$
X_2	Y_2	$1/5$
X_3	Y_3	$1/5$
X_4	Y_4	$1/5$
X_5	Y_5	$1/5$

Fitted tree $\hat{f}^1(x)$

Correctly predict all samples besides Y_3 and Y_5

$$\frac{1}{2}\log\frac{1 - Total\ Error}{Total\ Error} = \frac{1}{2}\log\frac{1 - 2/5}{2/5} = 0.088$$

Increase the sample weight for the sample that was incorrectly classified

$$New\ sample\ weight = sample\ weight \times \exp(Amount\ of\ stay)$$

$$New\ sample\ weight = \frac{1}{5} \times \exp(Amount\ of\ stay) = 0.2184$$

Decrease the sample weight for the sample that was correctly classified

$$New\ sample\ weight = sample\ weight \times \exp(-Amount\ of\ stay)$$

$$New\ sample\ weight = \frac{1}{5} \times \exp(-Amount\ of\ stay) = 0.1831$$

AdaBoost

Initial weight

X_1	Y_1	$1/5$
X_2	Y_2	$1/5$
X_3	Y_3	$1/5$
X_4	Y_4	$1/5$
X_5	Y_5	$1/5$

Update weight

New weight

X_1	Y_1	0.1831
X_2	Y_2	0.1831
X_3	Y_3	0.2184
X_4	Y_4	0.1831
X_5	Y_5	0.2184

Fitted tree $\hat{f}^1(x)$
Correctly predict all samples besides Y_3 and Y_5

$$Sum\ of\ the\ weights\ =\ 0.9862 \neq 1$$

AdaBoost

Initial weight

X_1	Y_1	1/5
X_2	Y_2	1/5
X_3	Y_3	1/5
X_4	Y_4	1/5
X_5	Y_5	1/5

Update weight

New weight

X_1	Y_1	0.1831/0.9862
X_2	Y_2	0.1831/0.9862
X_3	Y_3	0.2184/0.9862
X_4	Y_4	0.1831/0.9862
X_5	Y_5	0.2184/0.9862

…

Fitted tree $\hat{f}^1(x)$
Correctly predict all samples besides Y_3 and Y_5

Fitted tree $\hat{f}^2(x)$

Predict the most likely class: $\hat{f}(x) = Sign\left(\sum_{b=1}^{B} \lambda_b \hat{f}^b(x)\right)$, recall that $Y \in \{-1,1\}$, so is $\hat{f}^b(x)$

Lecture plan

- **Neural networks**

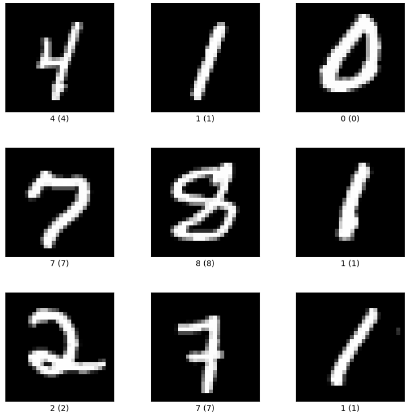
- Geoff Hinton on receiving Nobel prize for his work on laying the foundation of artificial neural networks

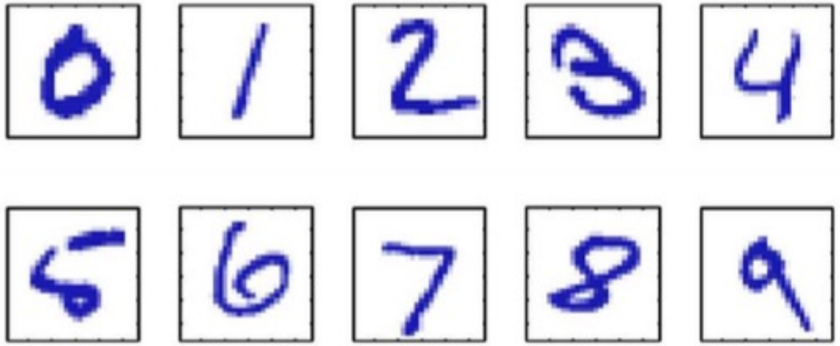
- https://www.youtube.com/watch?v=DNQ9YbyUNSQ

- https://www.youtube.com/watch?v=-icD_KmvnnM

Simplest problem: Handwritten digit recognition

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

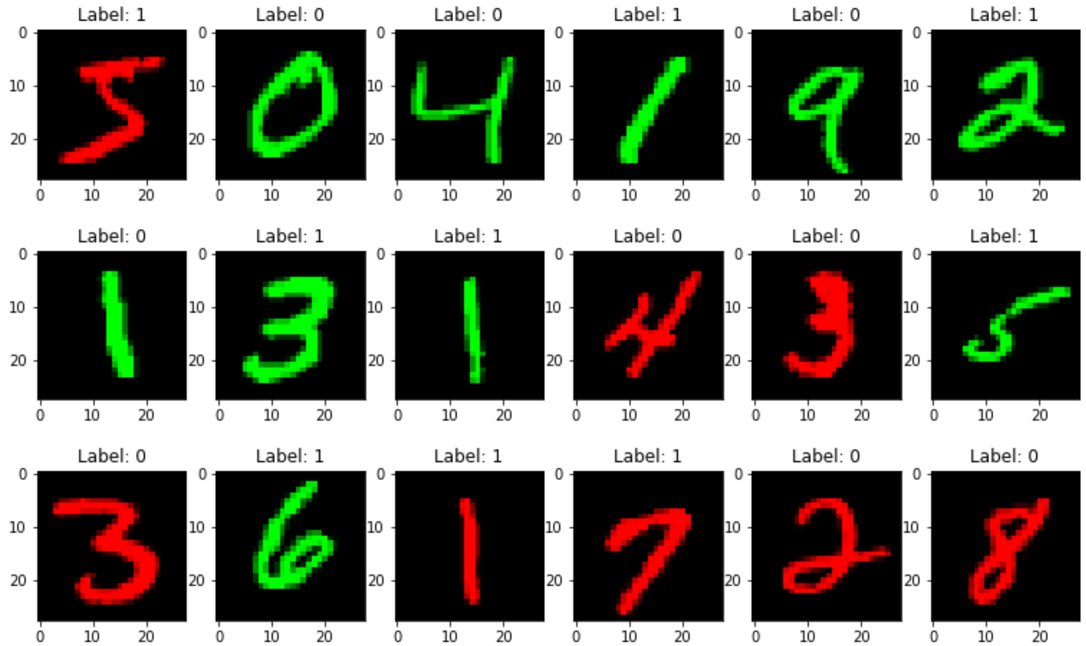


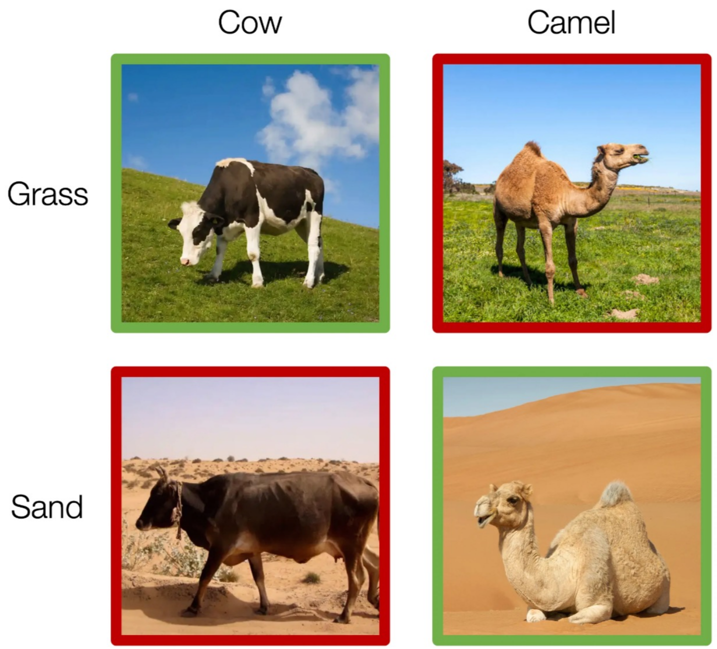


- **MNIST:** http://yann.lecun.com/exdb/mnist/
  - 50,000 handwritten digits for training; 5,000 for validation; 5,000 for testing

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

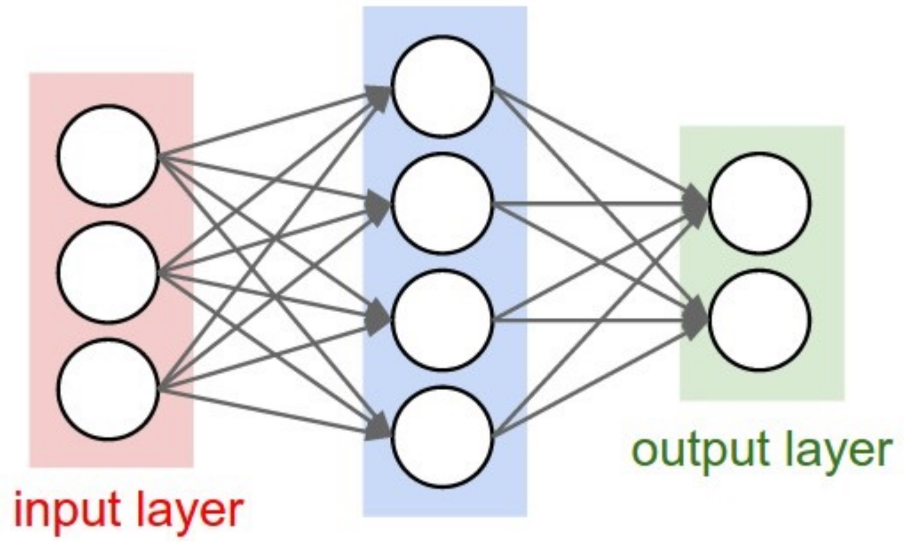
Housing numbers

- Street view house numbers: http://ufldl.stanford.edu/housenumbers/

- 73,257 digits for training; 26,032 digits for testing; 531,131 unlabeled digits
- Similar examples for car plates (e.g., highway tolls)

Feedforward neural networks

- Example of a feedforward neural network



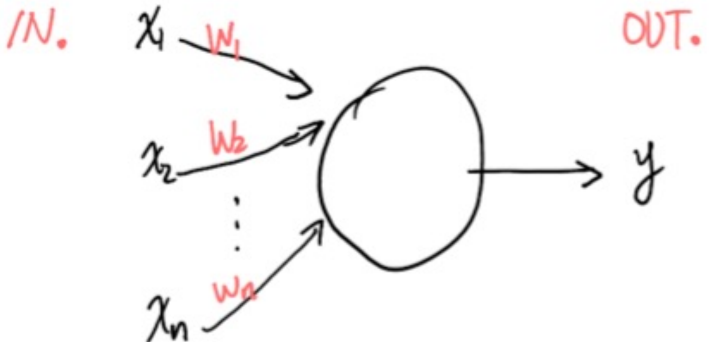
- This simple approach works well on MNIST and other handwritten digit recognition examples

Lecture plan

- **Artificial neuron: Perceptron, https://en.wikipedia.org/wiki/Perceptron**

An artificial neuron

- Perceptron is a type of artificial neuron



- Input: n real values $x_1, x_2, \dots, x_n$
- Weight parameters $w_1, w_2, \dots, w_n$ 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 brand-new restaurant that had just 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?"; $w_3 = -10$

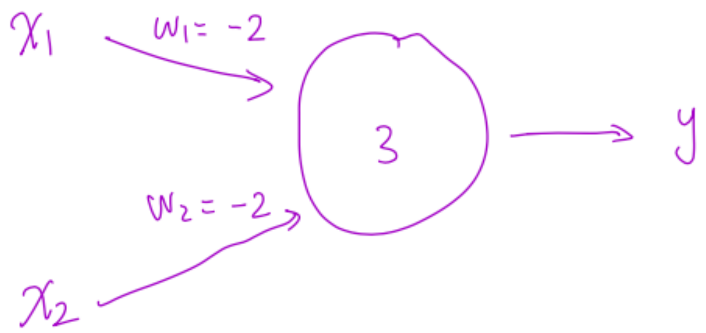
- Budget $\mathrm{b} = 40$
  - If $x_1 = 1, x_2 = 1, x_3 = 0$, then $y = 1$
  - If $x_1 = 0, x_2 = 1, x_3 = 1$, then $y = 1$
  - If $x_1 = 1, x_2 = 1, x_3 = 1$, then $y = 0$

Succinct 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

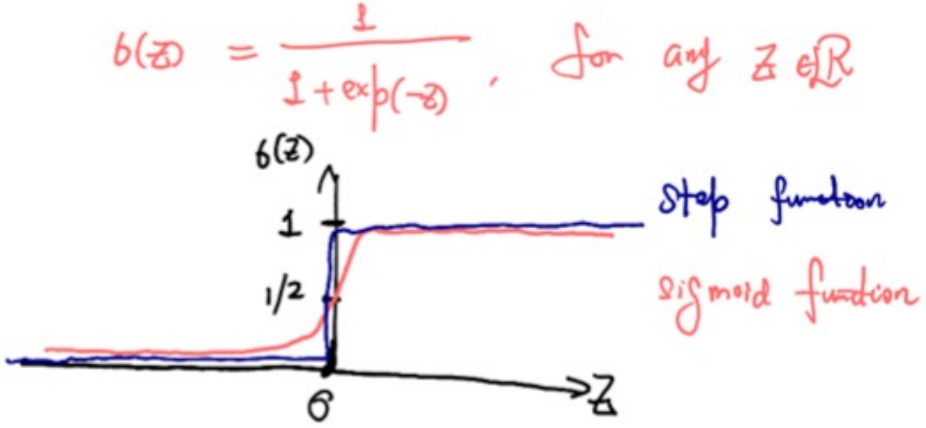
- Compute elementary logical functions
- **Example**: Use a perceptron to represent Negated AND



- 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

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

Sigmoid

- 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

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

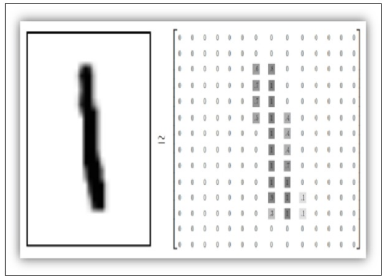
- Sigmoid neurons are differentiable: can run auto-differentiation in PyTorch or TensorFlow

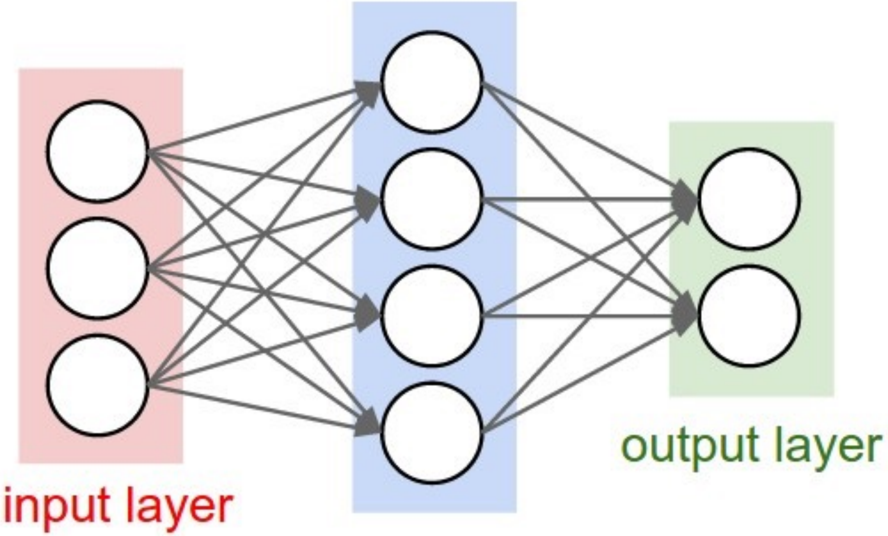
Lecture plan

- **Neural network architecture**

A closer look of every component

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



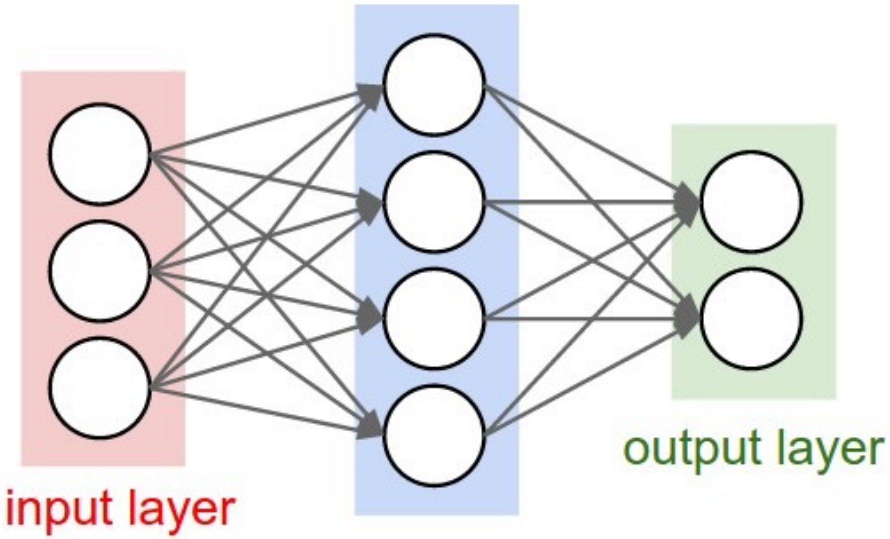


Prediction over $\{0,\textcolor{red}{\mathbf{1}},2,3,4,5,6,7,8,9\}$

hidden layer with four neurons

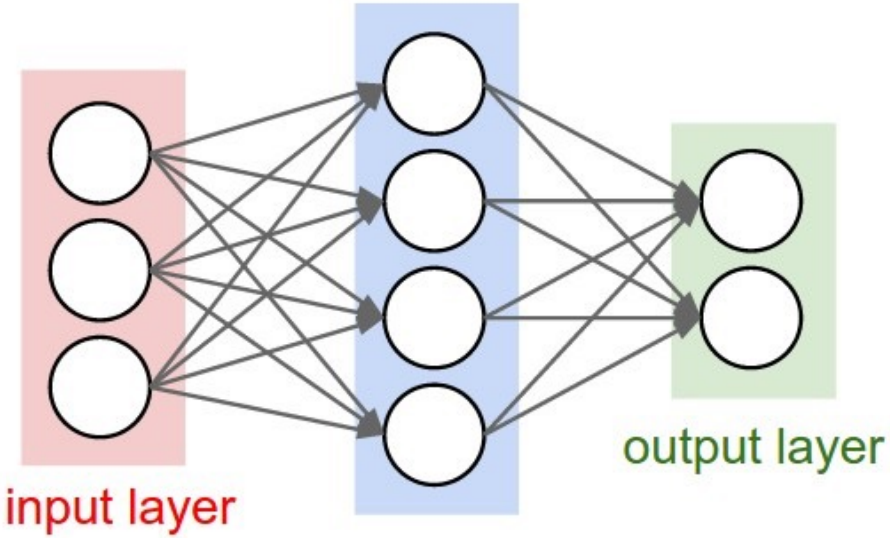
Design choices

- 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

Convolutional neural networks

- Number of parameters is often very large for modern neural networks
- Large parameter space comes with large model capacity

Deep networks

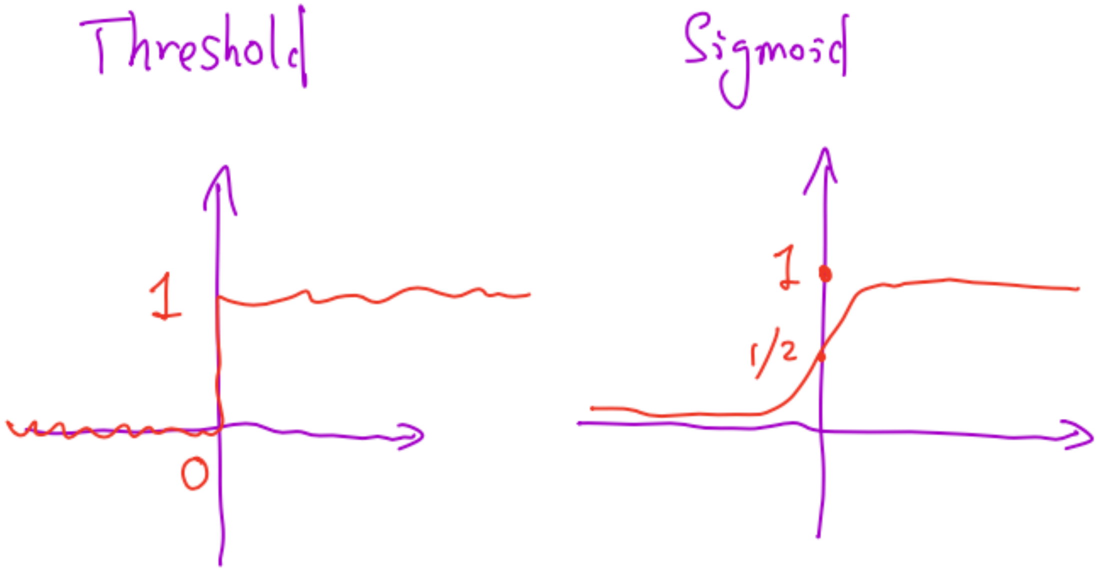
ConvNet	# Params
AlexNet	60M
VGG19	140M
ResNet-50	25M

≫ IMAGENET **1.5M**

- Number of parameters can be much higher than the number of labeled examples

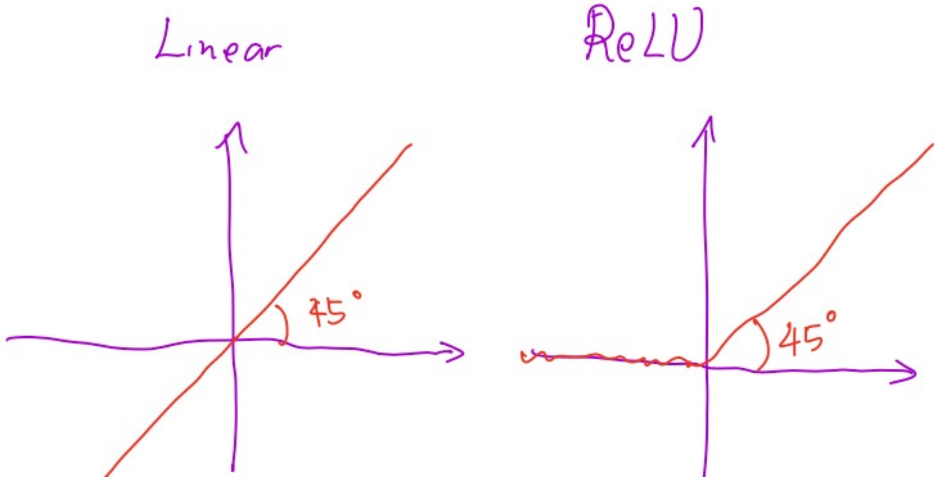
Design choices

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



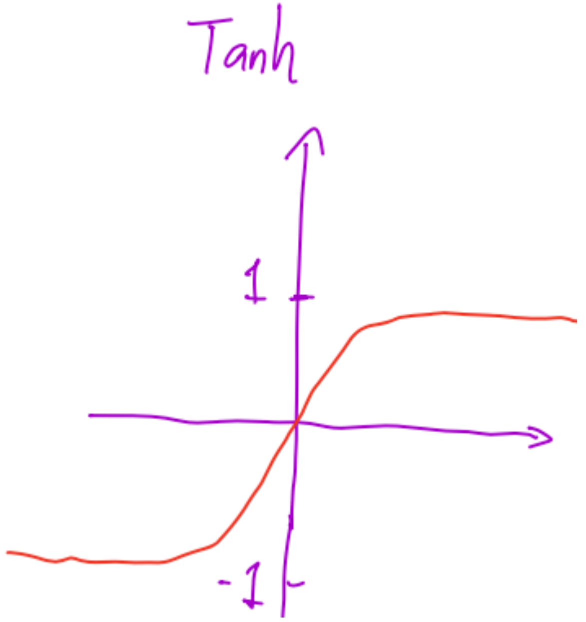
Design choices

- 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

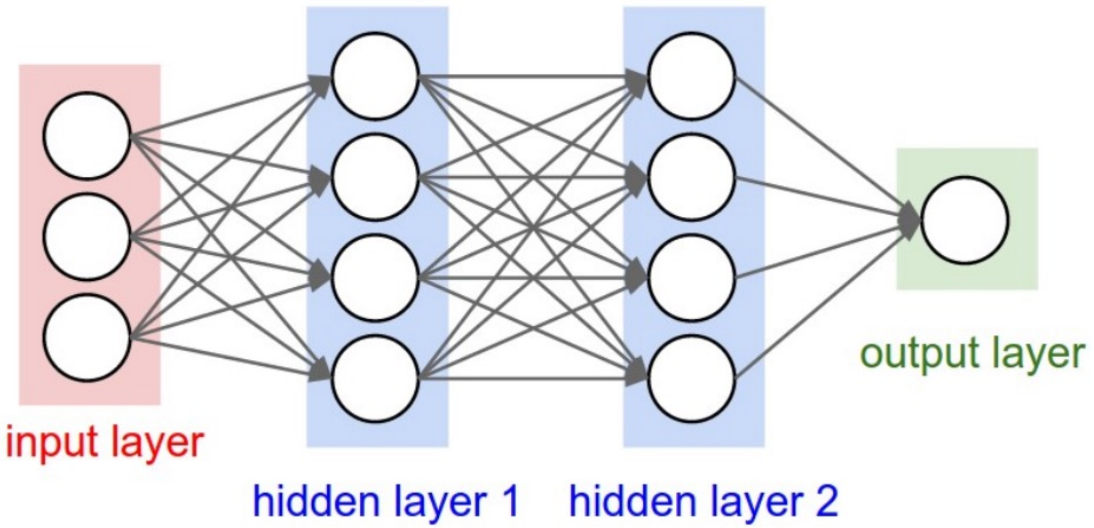
- Threshold function: $\sigma(z) = 0 \; if \; z \leq 0; \sigma(z) = 1 \; if \; z > 0$
- Sigmoid function: $\sigma(z) = \frac{1}{1+\exp(-z)}$
- Linear function: $\sigma(z) = z$
- Rectified linear units (ReLU): $\sigma(z) = \max(z, 0)$
- Tanh: $\sigma(z) = \frac{e^{2z}-1}{e^{2z}+1}$, similar to sigmoid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

Multi-layer neural networks

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



Notes

- 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

- **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

Cross-entropy loss

- Given a prediction for every label $y \in \{1,2,\ldots,k\}$, let u be this vector
- Softmax maps u into a probability distribution:

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

- Example: for MNIST, the label space is $\{0,1,2,\ldots,9\}$. A softmax output for **1** should look like [0.01, **0.9**, 0.01, 0.01, 0.01, 0.01, 0.01, 0.02, 0.01, 0.01]
- Illustrate the gradient of cross-entropy loss

PyTorch

CrossEntropyLoss = Negative Log Likelihood applied to SoftMax

- 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

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

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

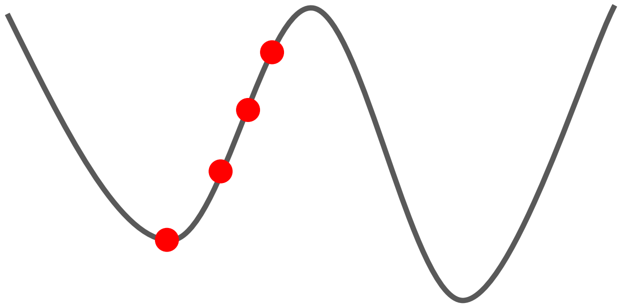
https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html

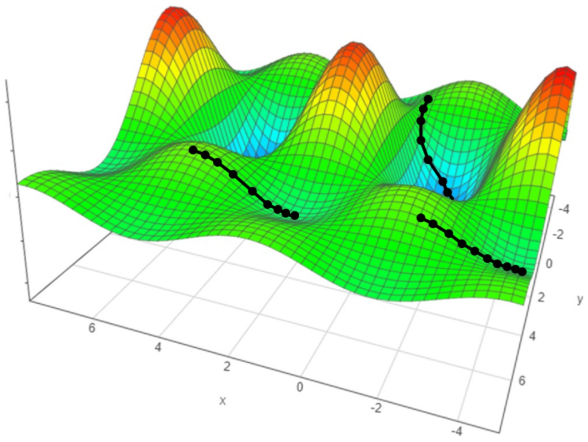
Gradient descent

- After setting up the loss $l(f(x_i), y_i)$, we set up an algorithm to minimize the empirical loss, measured as the averaged loss on the training set

$$\hat{L}(f_W) = \frac{1}{n} \sum_{1 \le i \le n} l(f_W(x_i), y_i)$$

- We use optimization algorithms like gradient descent that are quick to run





The gradient descent algorithm

- Let w_t be the parameters of a neural network
- Let f_{w_t} be the neural network
- Let $\nabla\hat{L}(f_{w_t})$ be the gradient of the training loss at w_t

- Let η be a learning rate parameter

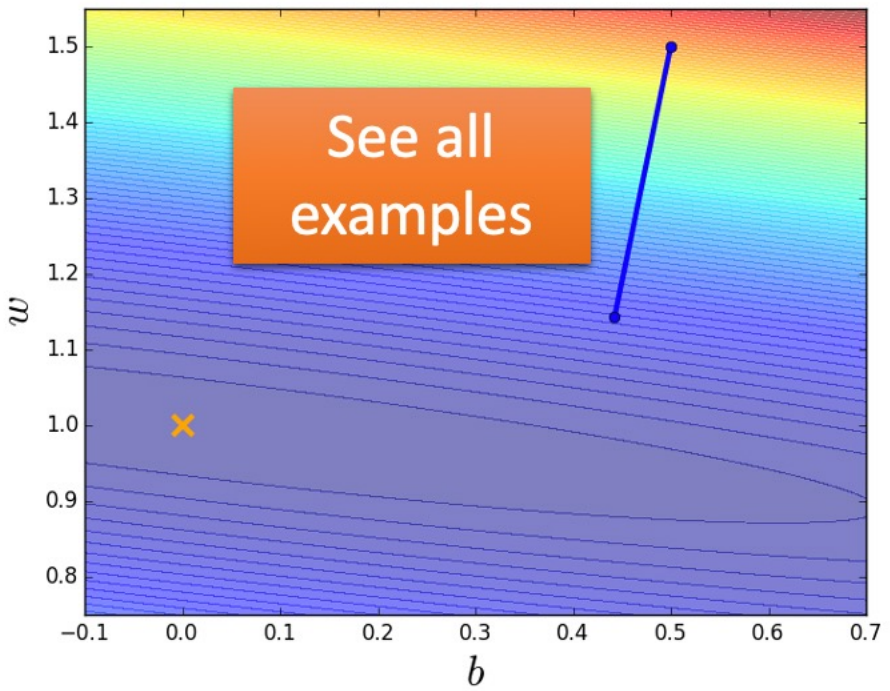
$$w_t \leftarrow w_t - \eta \cdot \nabla\hat{L}(f_{w_t})$$

Stochastic gradient descent

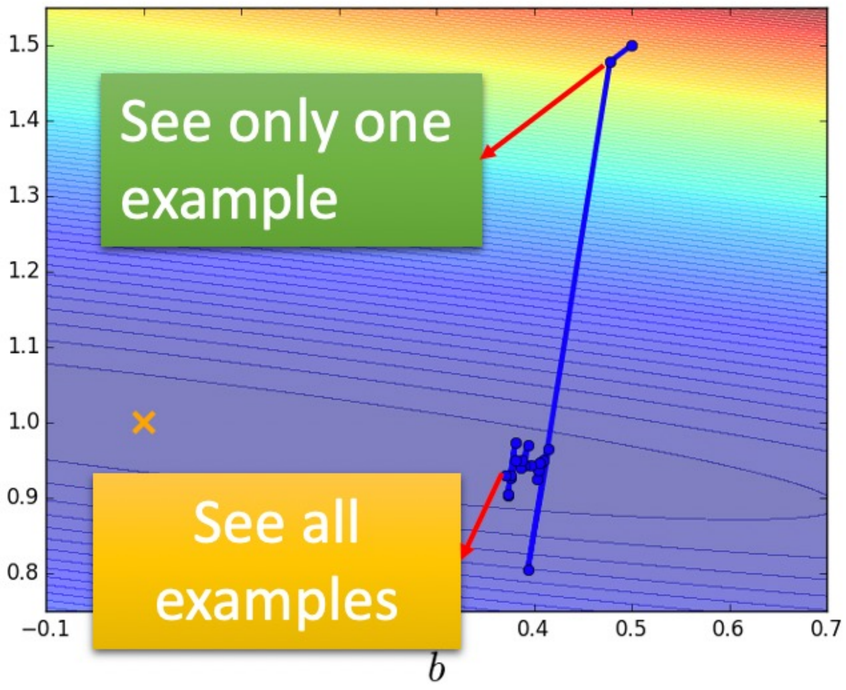
- **Motivation:** If the gradient on the first half is almost identi half
  - Mini-batch stochas

GD vs. SGD

- **Gradient Descent:** Update after seeing all examples

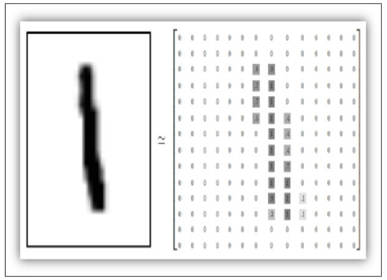


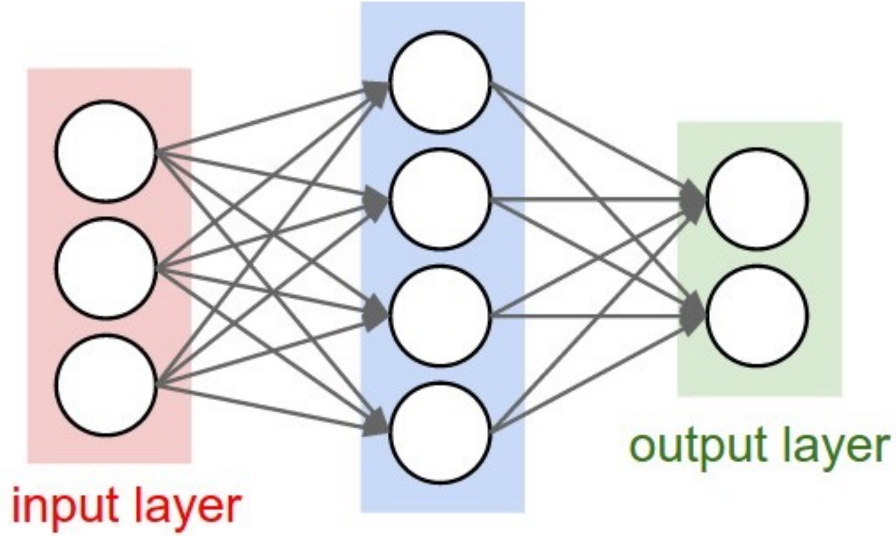
- **Stochastic Gradient Descent:** Update for each example



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]

Announcements

- Suggested reading: ISLP, Chapter 10: 10.1 and 10.2
- HW1 grading released: Questions or regrade requests, submit on gradescope or post a private note on piazza/canvas!
- HW2 due next Monday
- See course schedule here https://docs.google.com/spreadsheets/d/1AK1BuVMTe2jE0r8YhPKHjlPycJRUB6YsIzlIFyN73iA/edit?gid=0#gid=0