# Lecture 23: Neural Networks and Pretraining

Jacob Steinhardt

April 13, 2021

# Neural Networks

# Motivation

Recall linear regression / classification setup:

$$L(\beta) = \frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - \beta^\top x^{(i)})^2 \text{ (linear)}$$

$$L(\beta) = \frac{1}{n}\sum_{i=1}^{n} -\log \sigma((-1)^{y^{(i)}}\beta^\top x^{(i)}) \text{ (logistic)}$$

# Motivation

Recall linear regression / classification setup:

$$L(\beta) = \frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - \beta^\top x^{(i)})^2 \text{ (linear)}$$

$$L(\beta) = \frac{1}{n}\sum_{i=1}^{n} -\log\sigma((-1)^{y^{(i)}}\beta^\top x^{(i)}) \text{ (logistic)}$$

What if we want to learn more complex functions?
(E.g. true function not linear in $x$)

# Motivation

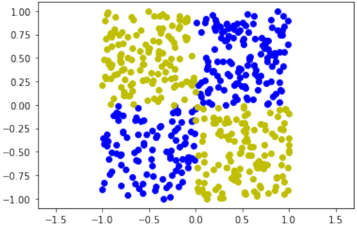
Recall linear regression / classification setup:

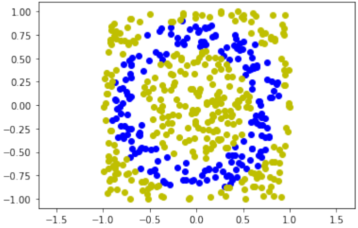
$$L(\beta) = \frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - \beta^\top \phi(x^{(i)}))^2 \text{ (linear)}$$

$$L(\beta) = \frac{1}{n}\sum_{i=1}^{n} -\log \sigma((-1)^{y^{(i)}} \beta^\top \phi(x^{(i)})) \text{ (logistic)}$$

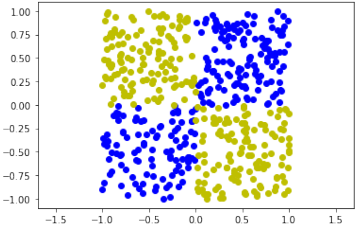
What if we want to learn more complex functions?
(E.g. true function not linear in $x$)

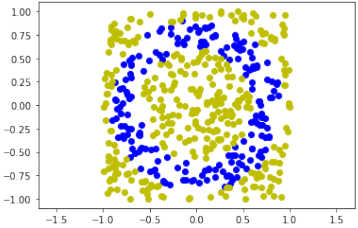
# Non-linear Examples



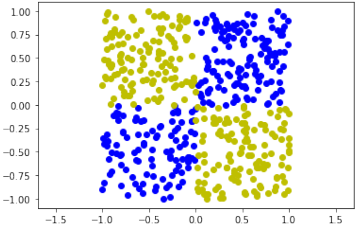
# Non-linear Examples

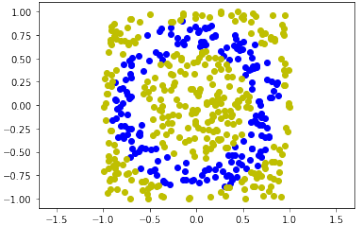



$\phi(x) = x_1 x_2$, $\phi(x) = |x_1^2 + x_2^2 - 0.6|$

# Non-linear Examples




$\phi(x) = x_1 x_2$, $\phi(x) = |x_1^2 + x_2^2 - 0.6|$

- This gets tedious.
- What if we can't think of good features ahead of time?

# Non-parametric modeling

**Non-parametric modeling:** define flexible function classes so we don't need to hand-engineer features.

# Non-parametric modeling

**Non-parametric modeling:** define flexible function classes so we don't need to hand-engineer features.

Many approaches:

- Random features
- Neural networks
- Kernels
- Decision trees

# Non-parametric modeling

**Non-parametric modeling:** define flexible function classes so we don't need to hand-engineer features.

Many approaches:

- Random features
- Neural networks
- Kernels
- Decision trees

Focus on first two for this lecture

# Random features

Input $x \in \mathbb{R}^d$, but can't think of good features function $\phi(x)$

# Random features

Input $x \in \mathbb{R}^d$, but can't think of good features function $\phi(x)$

Solution: make $\phi$ random but high-dimensional:

$$\phi(x) = \mathrm{sign}(Mx + b), \tag{1}$$

where $M \in \mathbb{R}^{d \times k}$ and $b \in \mathbb{R}^k$ are random vectors (chosen once at beginning).

# Random features

Input $x \in \mathbb{R}^d$, but can't think of good features function $\phi(x)$

Solution: make $\phi$ random but high-dimensional:

$$\phi(x) = \mathrm{sign}(Mx + b), \tag{1}$$

where $M \in \mathbb{R}^{d \times k}$ and $b \in \mathbb{R}^k$ are random vectors (chosen once at beginning).

Other features work too, e.g. $\cos(Mx + b)$, etc. Key points are randomness (good variation) and high dimensionality (usually $k > d$).

- Will show later this is (approximately) equivalent to kernel regression!

# Random features: Jupyter demo

[switch to notebook]

# Learned features

- Random features can be too crude
- What if features themselves are learnable?

# Learned features

- Random features can be too crude
- What if features themselves are learnable?

Two-layer neural network:

$$\phi(x) = \sigma(M_1 x + b_1),$$
$$p(y \mid x) = \sigma(M_2 \sigma(M_1 x + b_1) + b_2).$$

# Learned features

- Random features can be too crude
- What if features themselves are learnable?

Two-layer neural network:

$$\phi(x) = \sigma(M_1 x + b_1),$$
$$p(y \mid x) = \sigma(M_2 \sigma(M_1 x + b_1) + b_2).$$

Modern ML: iterate to many layers (and use different non-linearity $\sigma$, convolutional structure, etc.)

# Learned features: Jupyter demo

[switch to notebook]

# Fitting a neural network model

How do we actually fit $M$ and $b$?

Recall stochastic gradient descent: update parameters $w = (M_1, M_2, b_1, b_2)$ by following gradient of the loss $\nabla L(w)$:

$$w' \leftarrow w - \eta \nabla L(w)$$

How do we compute $\nabla L(w)$?

# Computing the gradient

[on board]

# Backpropagation and autodiffentiation

- Given any “computation graph”, we can write down derivatives recursively using the chain rule
- Then solve using dynamic programming!
- This is called backpropagation or autodifferentiation, key idea in Pytorch and other libraries

# Backprop in pytorch

[Jupyter demo]

# Pre-training

# Motivation

- Suppose we want to train a classifier to predict the political slant of news
- Common situation:
  - Lots of unlabeled data (all text on internet)
  - Few labeled data (hand-label 1000 random articles)
  - New instances might be OOD (news changes over time)
- How do we handle all the unlabeled data?
  - First *pretrain* on very large amount of (possibly unlabeled) data
  - Then *finetune* on smaller amount of labeled, task-specific data

# Pre-training: Examples

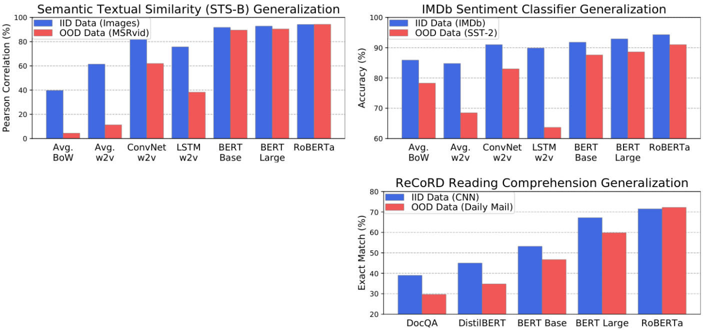
- Images: pretrain to predict Instagram tags (3.5B images), fine-tune on ImageNet (1M images)
- Images: pretrain on ImageNet (1M images), fine-tune on CIFAR-10 (50K images)
- Text: pretrain on Wikipedia (2.5B words) + BookCorpus (0.8B words), fine-tune on [sentiment classification, entailment, etc.]

Largest language models pretrained on over 400B tokens!

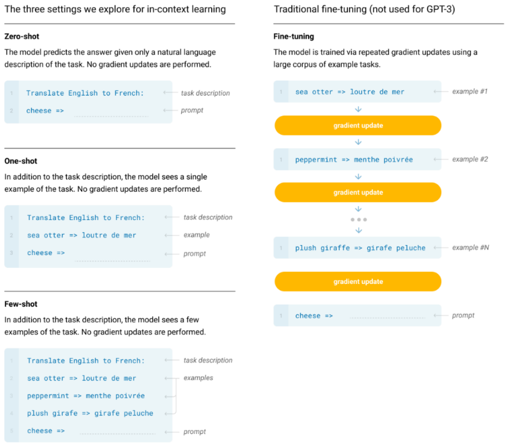
# Pre-training: Details

[on board]

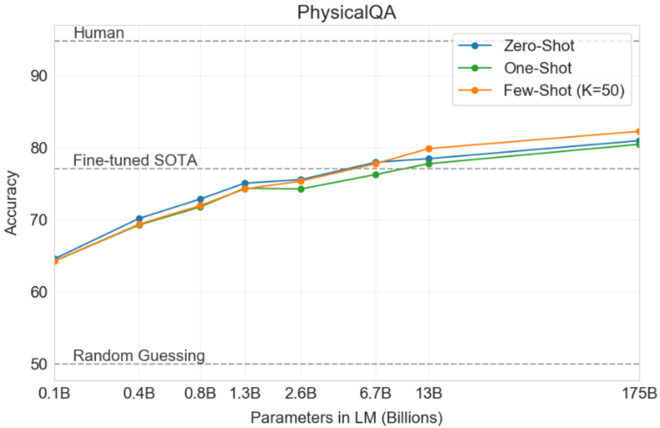
# Accuracy and Robustness



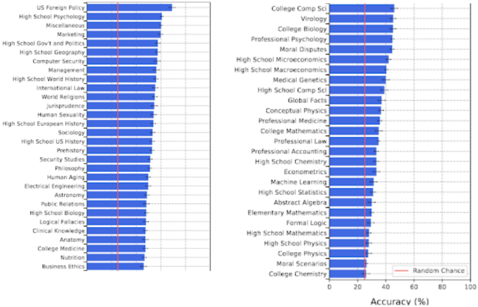
# Zero-shot Learning



# Few-shot Accuracy (I)



# Few-shot Accuracy (II)



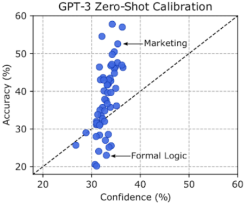




Figure 8: GPT-3's confidence is a poor estimator of its accuracy and can be off by up to 24%.