CS 839: Foundation Models
ML Mini-Review

Fred Sala

University of Wisconsin-Madison

Sept. 9, 2025

Announcements

- **OH:** Thurs 2:30-4:00 PM in Morgridge 5514
- **Resources**
  - https://mlstory.org/ : fun book by Hardt and Recht
- Class roadmap:

Tuesday Sept. 9	ML Mini-Review
Thursday Sept. 11	Architectures I: Transformers & Attention
Tuesday Sept. 16	Architectures II: Subquadratic Architectures
Thursday Sept. 18	Language Models I
Tuesday Sept. 23	Language Models II

Mostly Language Model

Outline

- **General Supervised Learning Review**
  - Features, labels, hypothesis classes, training, generalization
- **Neural Networks**
  - Perceptrons, MLPs, training and backprop, CNNs, brief review of RNNs and LSTMs, data augmentation
- **Self-Supervised Learning**
  - Getting representations, pretext tasks, using representations

Supervised Learning: Formal Setup

Problem setting

- Set of possible instances $\mathcal{X}$
- Unknown *target function* $f : \mathcal{X} \to \mathcal{Y}$
- Set of *models* (a.k.a. *hypotheses*): $\mathcal{H} = \{h | h : \mathcal{X} \to \mathcal{Y}\}$

Get

- Training set of instances for unknown target function,

$$(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(n)}, y^{(n)})$$


safe


poisonous


safe

Supervised Learning: Objects

Three types of sets

- Input space, output space, hypothesis class

$$\mathcal{X}, \mathcal{Y}, \mathcal{H}$$

- **Examples**:
  - Input space: feature vectors $\mathcal{X} \subseteq \mathbb{R}^d$

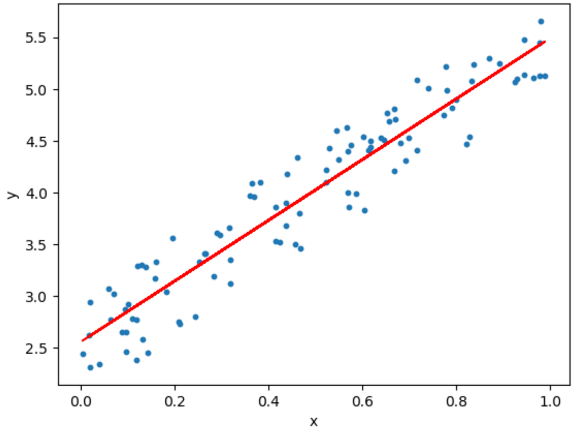
  - Output space:
    - **Binary** $\mathcal{Y} = \{-1, +1\}$ safe poisonous
    - **Continuous** $\mathcal{Y} \subseteq \mathbb{R}$ 13.23°

Output Space: Classification vs. Regression

Choices of $\mathcal{Y}$ have special names:

- Discrete: "**classification**". The elements of $\mathcal{Y}$ are **classes**
  - Note: doesn't have to be binary

- Continuous: "**regression**"
  - Example: linear regression



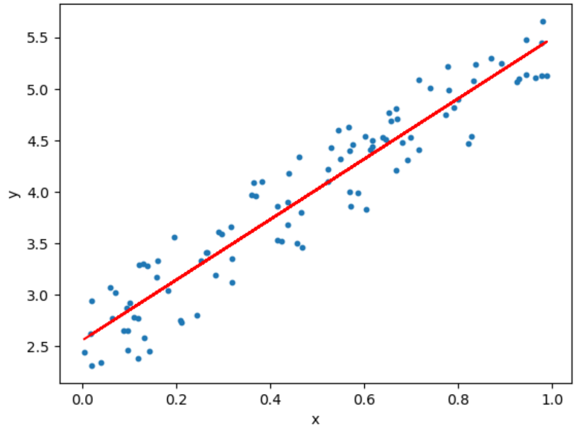
- There are other types…

Hypothesis Class

We talked about $\mathcal{X}, \mathcal{Y}$ what about $\mathcal{H}$?

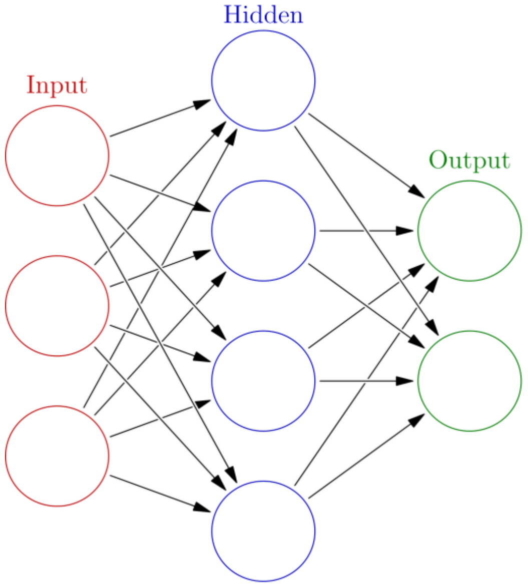
- Pick specific class of models. Ex: **linear models:**

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_d x_d$$



- Ex: **feedforward neural networks**

$$f^{(k)}(x) = \sigma(W_k^T f^{(k-1)}(x)))$$



- **Parameters:** θ, w.

Wikipedia

SL: Training & Generalization

Goal: model *h* that best approximates *f*

- One way: empirical risk minimization (ERM)

$$\hat{f} = \arg\min_{h \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} \ell(h(x^{(i)}), y^{(i)})$$

Model prediction

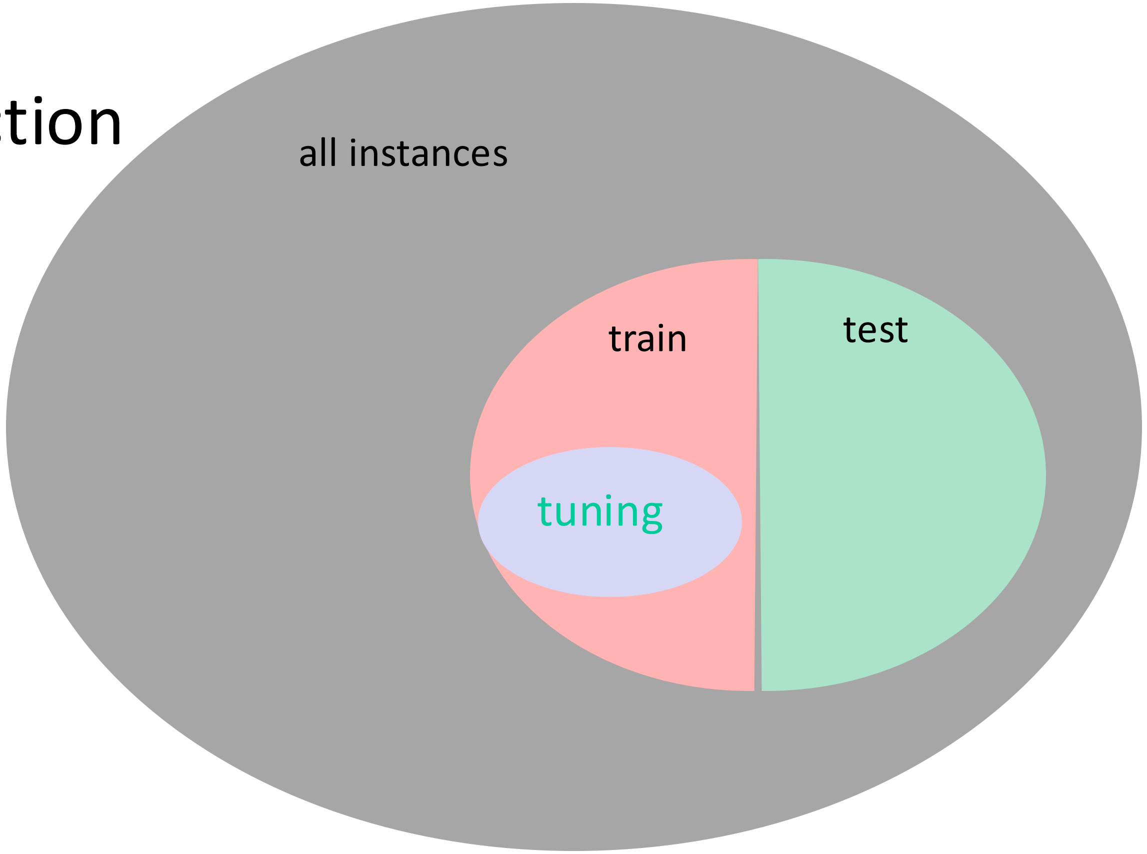
Hypothesis Class

Loss function (how far are we)?

- Generalization?

Evaluation: Validation and Test Sets

- A *validation set* (a.k.a. *tuning set*) is
  - Not used for primary training process, used to select among models
- A *test set*
  - Not used for training or selection
  - Compute metrics



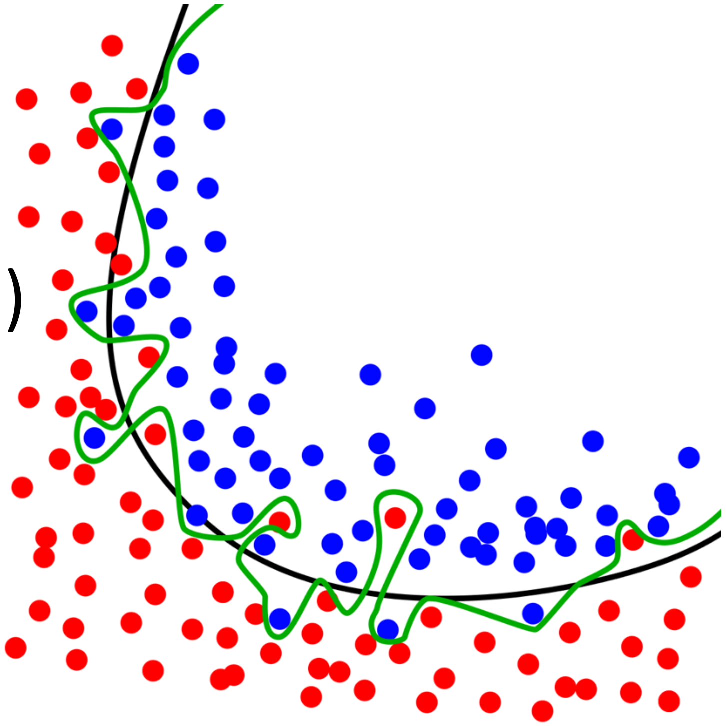
Overfitting

Notation: error of model *h* over

- training data: $\text{error}_{\mathrm{D}}(h)$
- entire distribution of data: $\text{error}_{\mathcal{D}}(h)$

Model *h* **overfits** training data if it has

- a low error on the training data (low $\text{error}_{\mathrm{D}}(h)$)
- high error on the entire distribution (high $\text{error}_{\mathcal{D}}(h)$)



Wikipedia

Beyond Accuracy: Confusion Matrices

- How can we understand what types of mistakes a learned model makes?

task: activity recognition from video

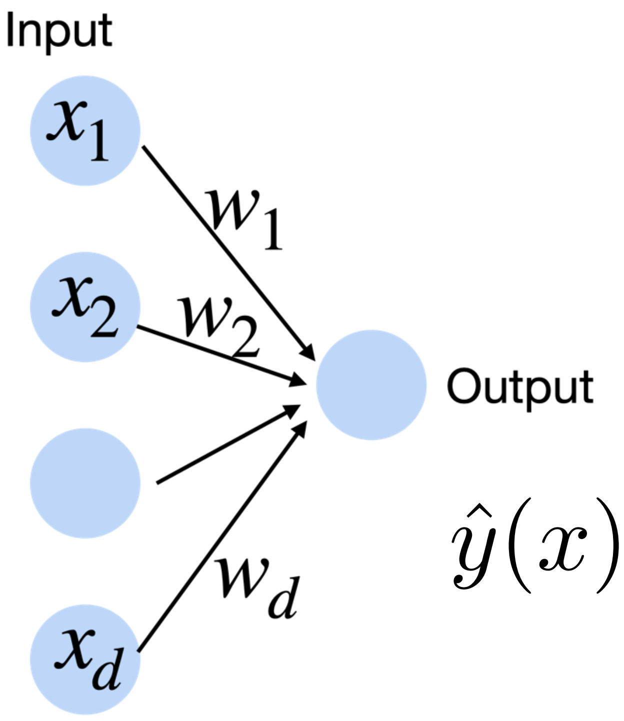
actual class

	bend	jack	jump	pjump	run	side	skip	walk	wave1	wave2
bend	100	0	0	0	0	0	0	0	0	0
jack	0	100	0	0	0	0	0	0	0	0
jump	0	0	89	0	0	0	11	0	0	0
pjump	0	0	0	100	0	0	0	0	0	0
run	0	0	0	0	89	0	11	0	0	0
side	0	0	0	0	0	100	0	0	0	0
skip	0	0	0	0	0	0	100	0	0	0
walk	0	0	0	0	0	0	0	100	0	0
wave1	0	0	0	0	0	0	0	0	67	33
wave2	0	0	0	0	0	0	0	0	0	100

predicted class

Break & Questions

Perceptron: Simple Network

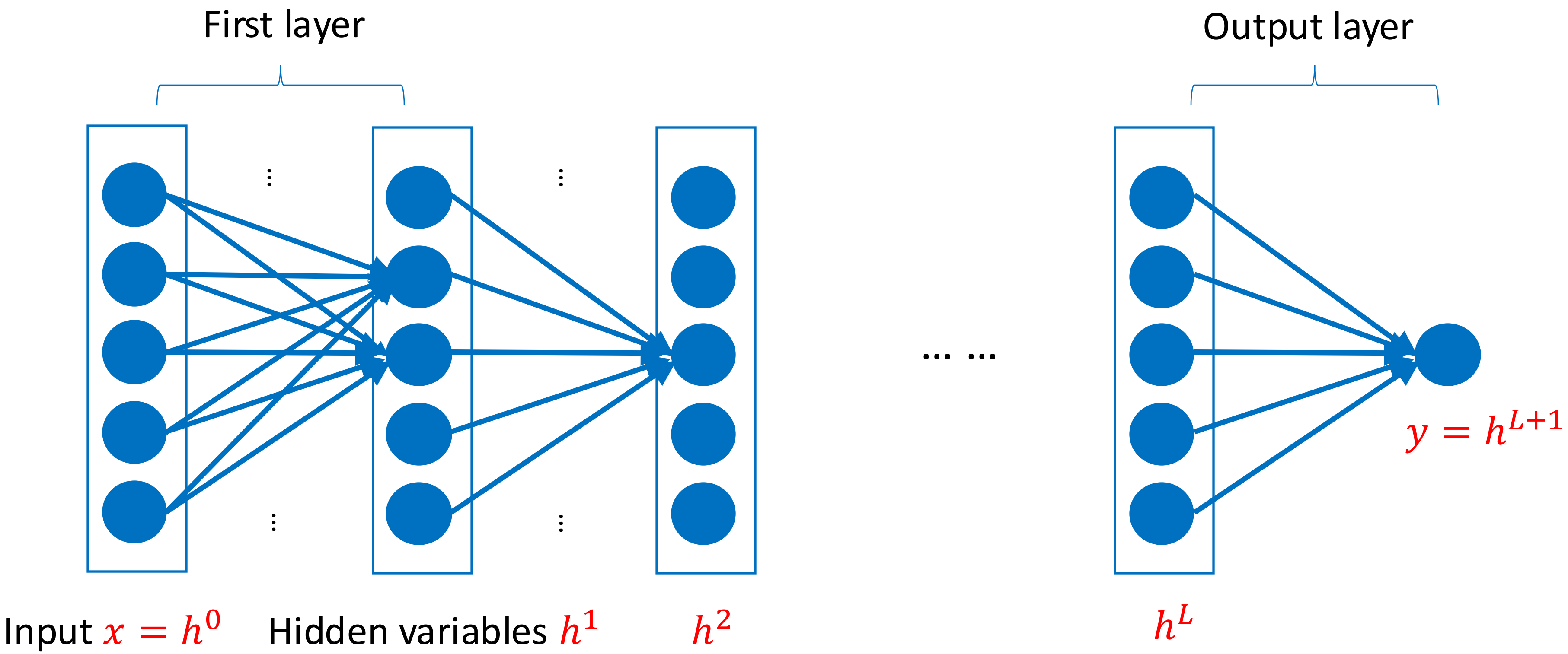


$$\hat{y}(x) = \begin{cases} 1 & w^T x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

[McCulloch & Pitts, **1943**; Rosenblatt, **1959**; Widrow & Hoff, **1960**]

Neural Networks: Multilayer Perceptrons

An $(L+1)$-layer network

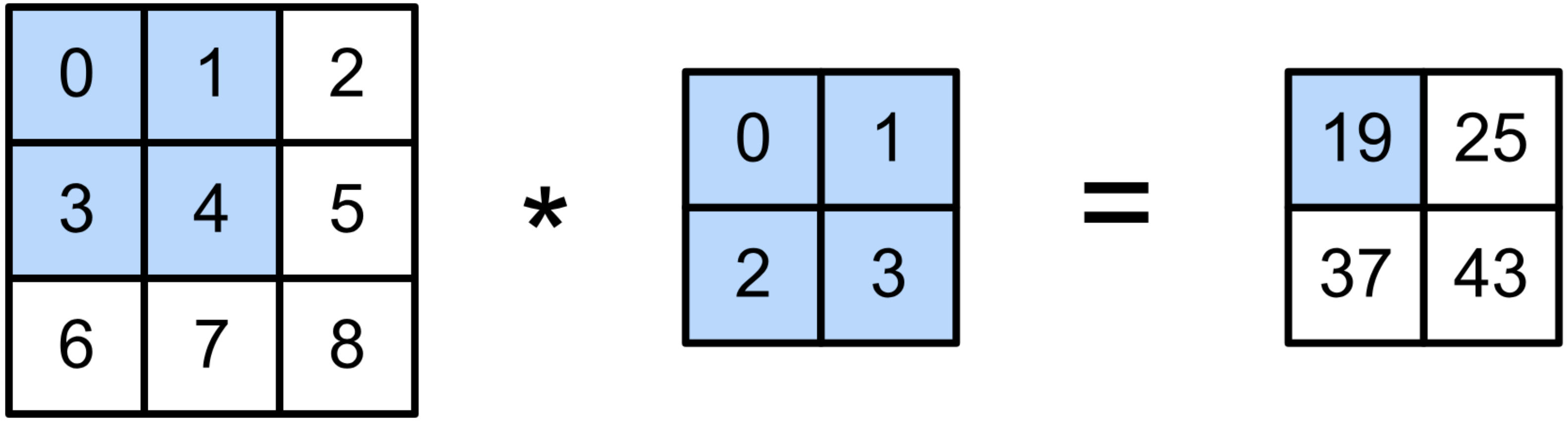


Training Neural Networks

- Algorithm:
  - Get $$D = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\}$$
  - Initialize weights
  - Until stopping criteria met,
    - For each training point $(x^{(i)}, y^{(i)})$
    - Compute: $f_{\text{network}}(x^{(d)})$ ⟵ **Forward Pass**
    - Compute gradient: $\nabla L^{(i)}(w) = \left[\frac{\partial L^{(d)}}{\partial w_0}, \frac{\partial L^{(d)}}{\partial w_1}, \ldots, \frac{\partial L^{(d)}}{\partial w_m}\right]^T$ ⟵ **Backward Pass**
    - Update weights: $$w \leftarrow w - \alpha \nabla L^{(i)}(w)$$

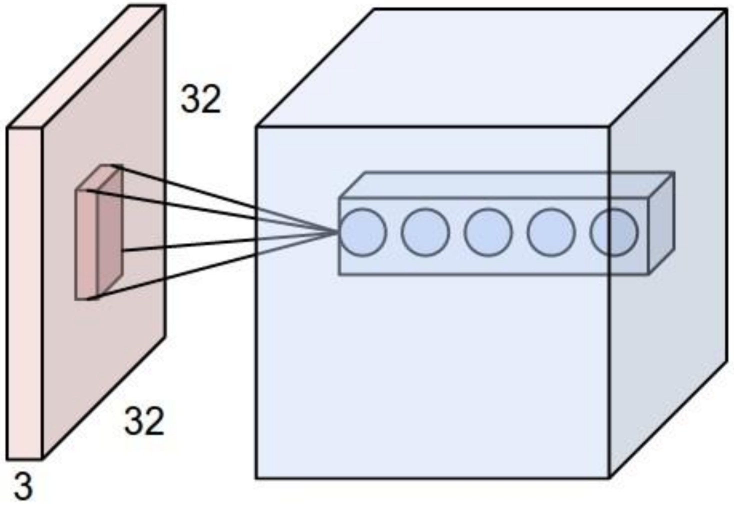
Neural Networks: Convolution Layers

- Notation:
  - X: n_h x n_w input matrix
  - W: k_h x k_w kernel matrix
  - b : bias (a scalar)
  - Y: () x () output matrix
- As usual *W*, *b* are learnable parameters



Neural Networks: Convolution NNs

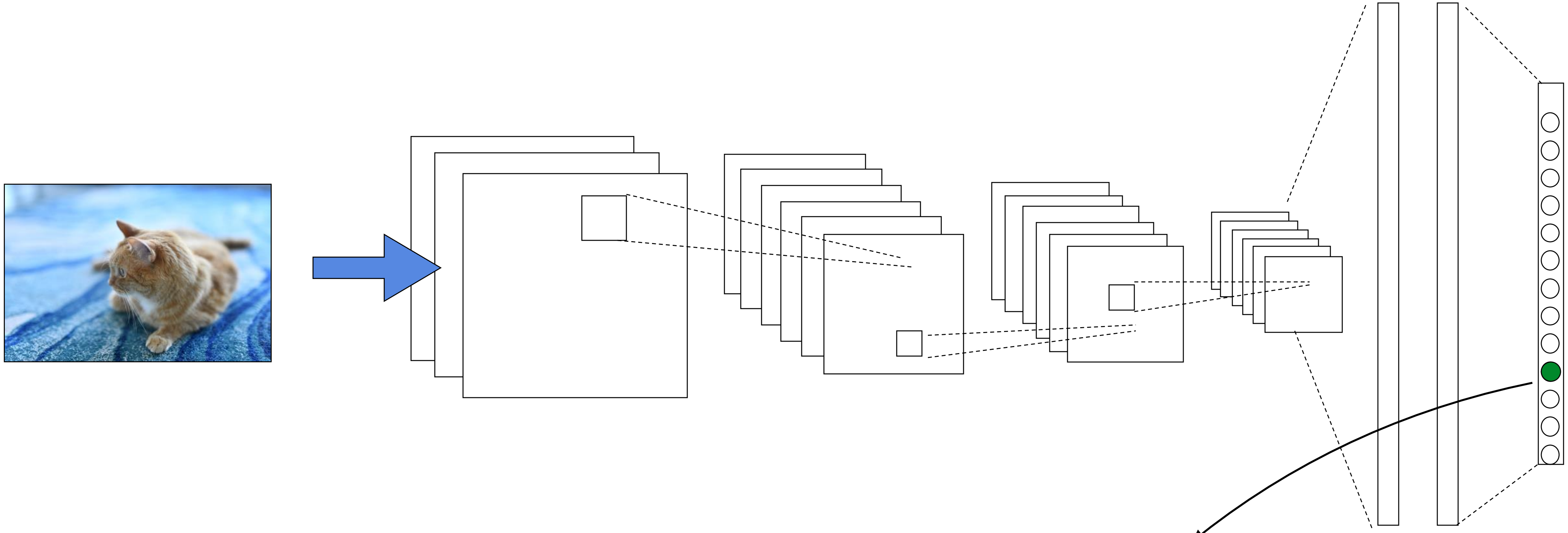
- Properties
  - Input: volume c_i x n_h x n_w (channels x height x width)
  - Hyperparameters: # of kernels/filters c_o, size k_h x k_w, stride s_h x s_w, zero padding p_h x p_w
  - Output: volume c_o x m_h x m_w (channels x height x width)
  - Parameters: k_h x k_w x c_i per filter, total $(k_h$ x k_w x $c_i)$ x c_o



Stanford CS 231n

Training a CNN

- Q: so we have a bunch of layers. How do we train?
- A: same as before. Apply softmax at the end, use backprop.

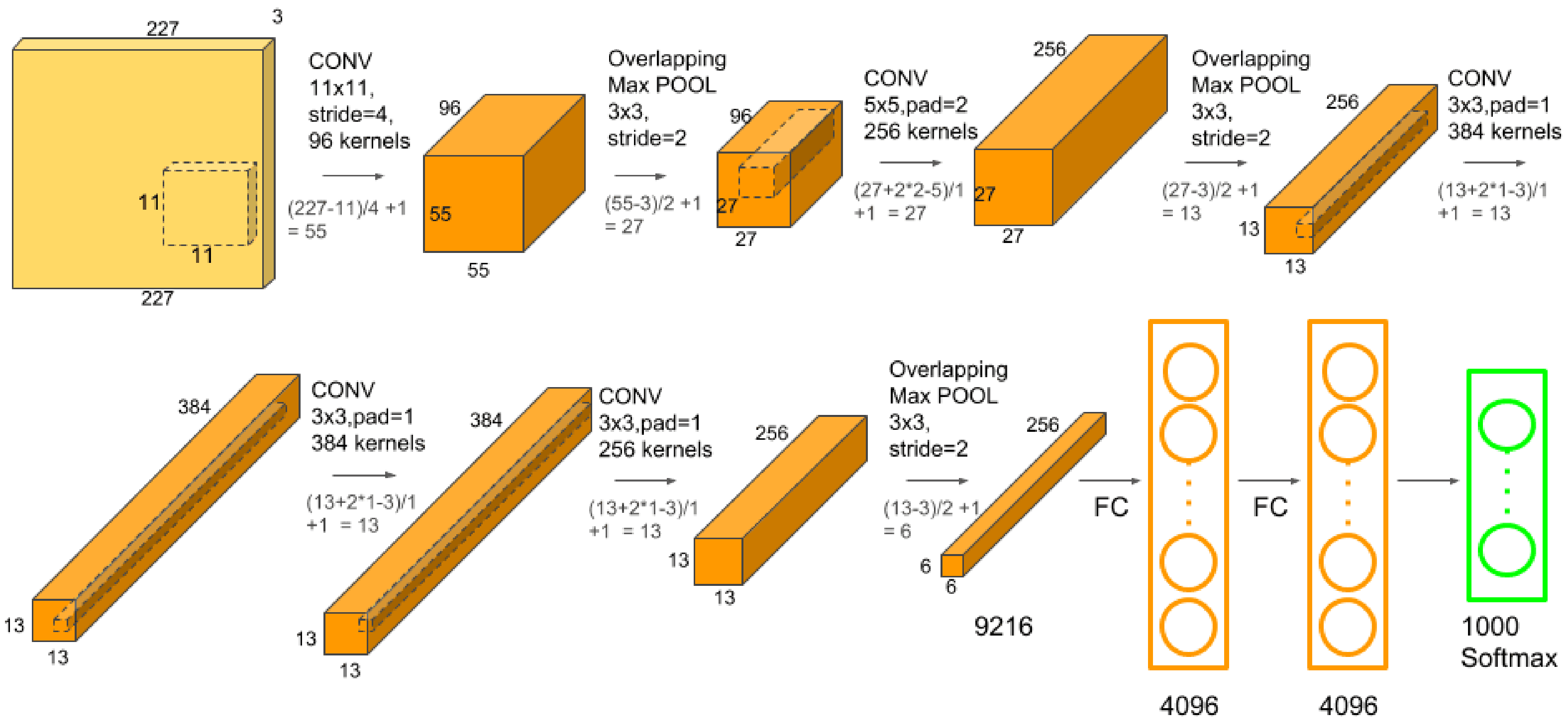


$$p_i(\boldsymbol{x}) = \frac{\exp\left(f_i(\boldsymbol{x})\right)}{\sum_{j=1}^{N} \exp\left(f_j(\boldsymbol{x})\right)},$$

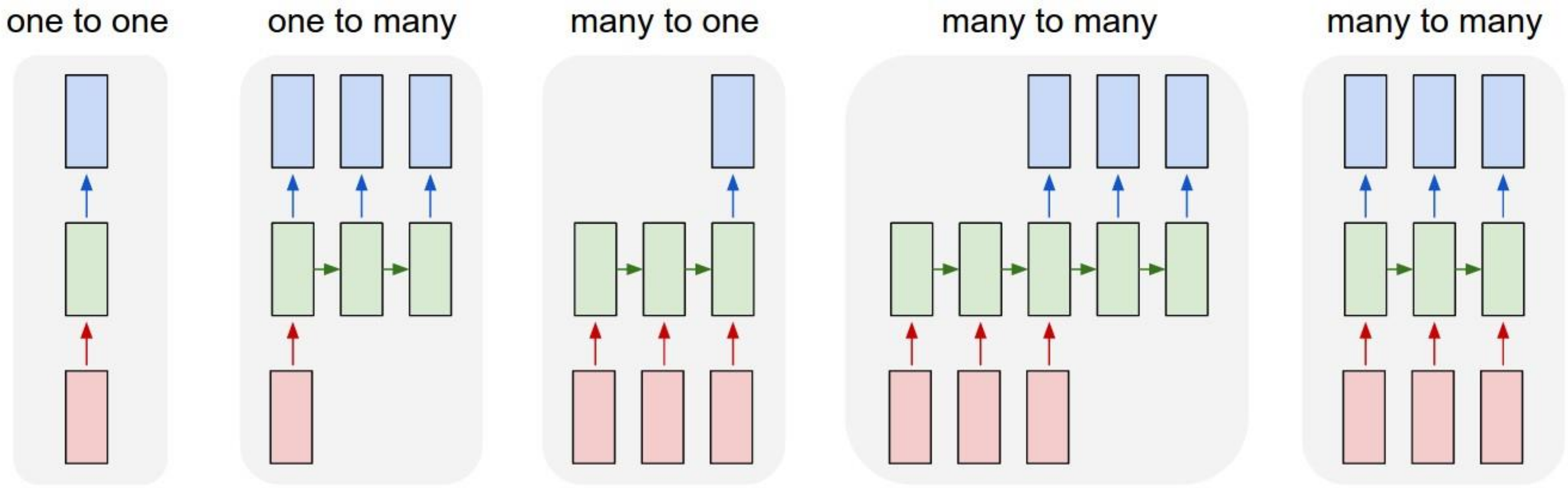
softmax

CNN Architectures: AlexNet

- First of the major advancements: AlexNet
- Wins 2012 ImageNet competition
- Major trends: deeper, bigger LeNet



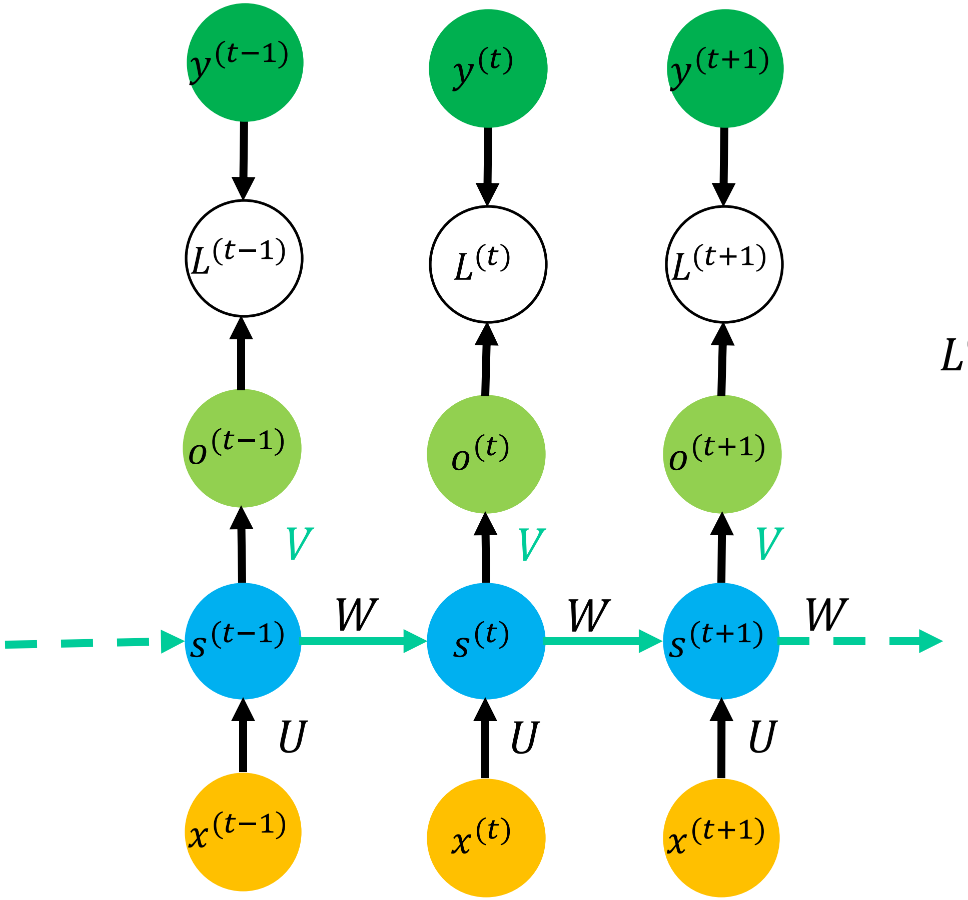
Tasks We Can Handle with NNs?



- Mostly talked about (1) so far
  - Others: need a new kind of model

Neural Networks: Simple RNNs

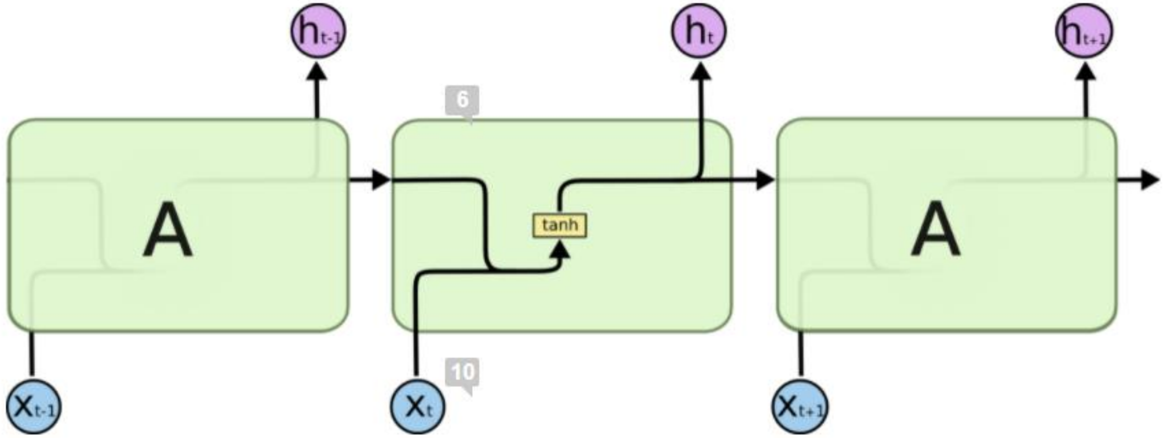
- Classical RNN variant:



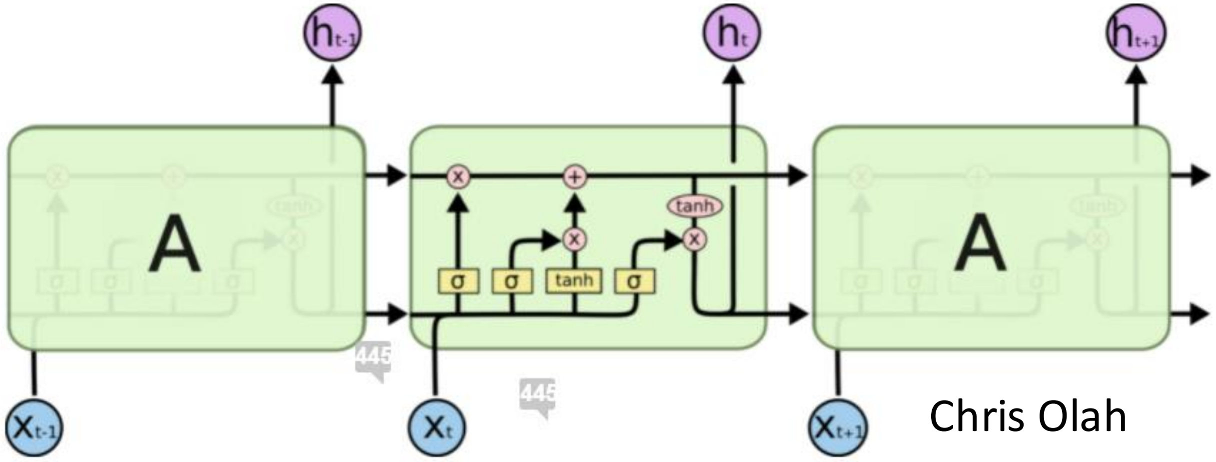
$$a^{(t)} = b + Ws^{(t-1)} + Ux^{(t)}$$
$$s^{(t)} = \tanh\left(a^{(t)}\right)$$
$$o^{(t)} = c + Vs^{(t)}$$
$$\hat{y}^{(t)} = \mathrm{softmax}\left(o^{(t)}\right)$$
$$L^{(t)} = \mathrm{CrossEntropy}(y^{(t)}, \hat{y}^{(t)})$$

Neural Networks: LSTMs

- RNN: can write structure as:



- Long Short-Term Memory: deals with problem. Cell:

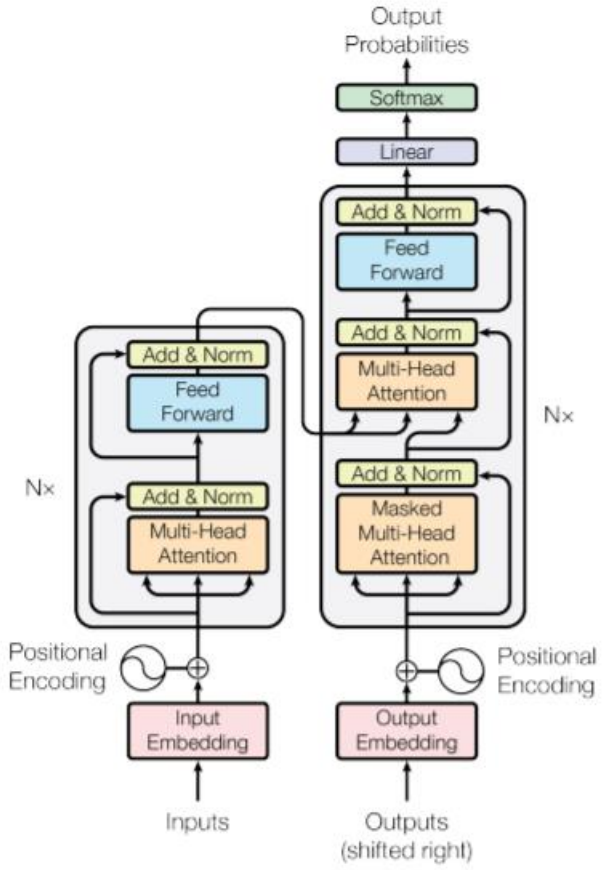


Chris Olah

Neural Networks: Transformers

- Initial goal for an architecture: **encoder-decoder**
  - Get **rid of recurrence**
  - Replace with **self-attention**

- Architecture
  - The famous picture you've seen
  - Centered on self-attention blocks



Vaswani et al. '17

Data Augmentation

Augmentation: transform + add new samples to dataset

- Transformations: based on domain
- Idea: build **invariances** into the model
  - **Ex**: if all images have same alignment, model learns to use it
- Keep the label the same!

Data Augmentation: Examples

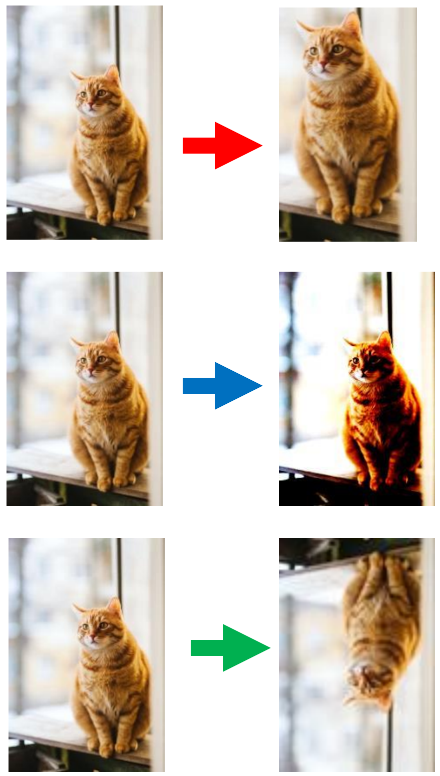
Examples of transformations for images

- **Crop** (and zoom)
- **Color** (change contrast/brightness)
- **Rotations+** (translate, stretch, shear, etc)

Many more possibilities. Combine as well!

Q: how to deal with this at **test time**?

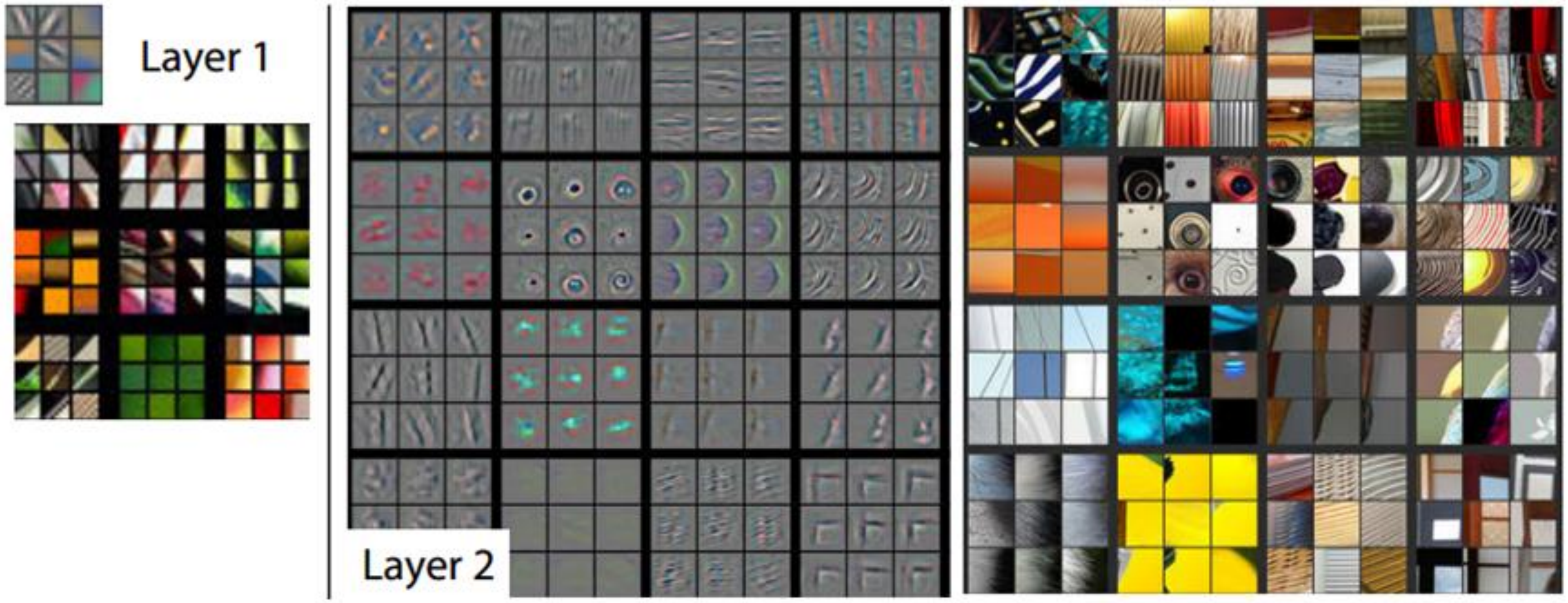
- A: transform, test, average



Break & Questions

Representations

- Basic idea in ML is to discover useful representations
  - I.e., higher level features that are discriminative
  - These are not necessarily present in raw data…



Visualizations of Layer 1 and 2. Each layer illustrates 2 pictures, one which shows the filters themselves and one that shows what part of the image are most strongly activated by the given filter. For example, in the space labled Layer 2, we have representations of the 16 different filters (on the left)

Desphande

Where to Get Representations?

- Deep learning:
  - Automatically obtain good features, but
  - **Downside**: Need lots of labeled data

- Pre-trained models:
  - E.g., ResNets trained on ImageNet. Use last layer (pre-prediction)
  - **Downside**: pre-trained task may not match our goal task

- Generative model encoders:
  - **Downside**: may not relate to semantics we care about

Representations from **Self Supervision**

- There's lots of information in our dataset already
  - Of course, specific to our task

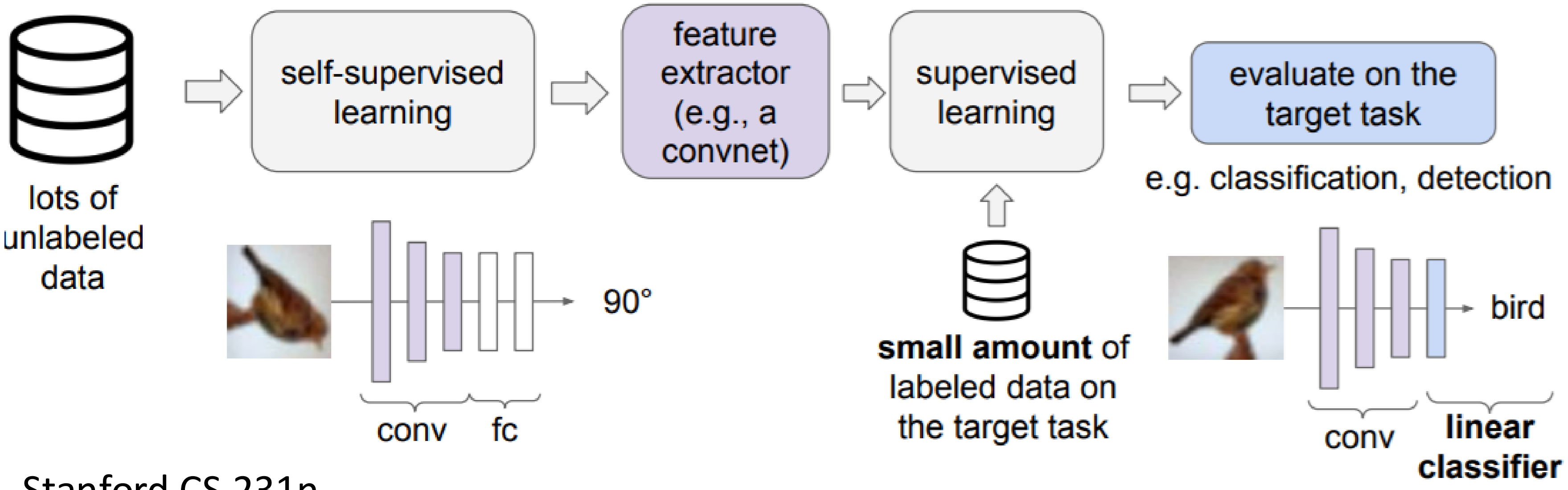
- Need to create tasks from unlabeled data: "Pretext tasks"
  - Ex: predict stuff you already know

image completion | rotation prediction | "jigsaw puzzle" | colorization

Stanford CS 231n

Using the Representations

- Don’t care specifically about our performance on self-task
- Use the learned network as a feature extractor
- Once we have labels for a particular task, train
  - A small amount of data



Stanford CS 231n

Terminology: Generative vs. Discriminative

Need a few terms to be re-used during class

- **Discriminative** model
  - Directly predict label $h(x) = y$ or compute $h(x) = p(y|x)$
  - Canonical example: **logistic regression**

$$P_\theta(y = 1|x) = \sigma(\theta^T x) = \frac{1}{1 + \exp(-\theta^T x)}$$

Terminology: Generative vs. Discriminative

Need a few terms to be re-used during class

- **Generative** model
  - Model $h(x,y) = p(x,y)$ or $h(x) = p(x)$. Can be unsupervised
  - Canonical example: **naïve Bayes**

$$P(X_1, \ldots, X_K, Y) = P(X_1, \ldots, X_K | Y) P(Y)$$
$$= \left(\prod_{k=1}^{K} P(X_k | Y) \right) P(Y)$$

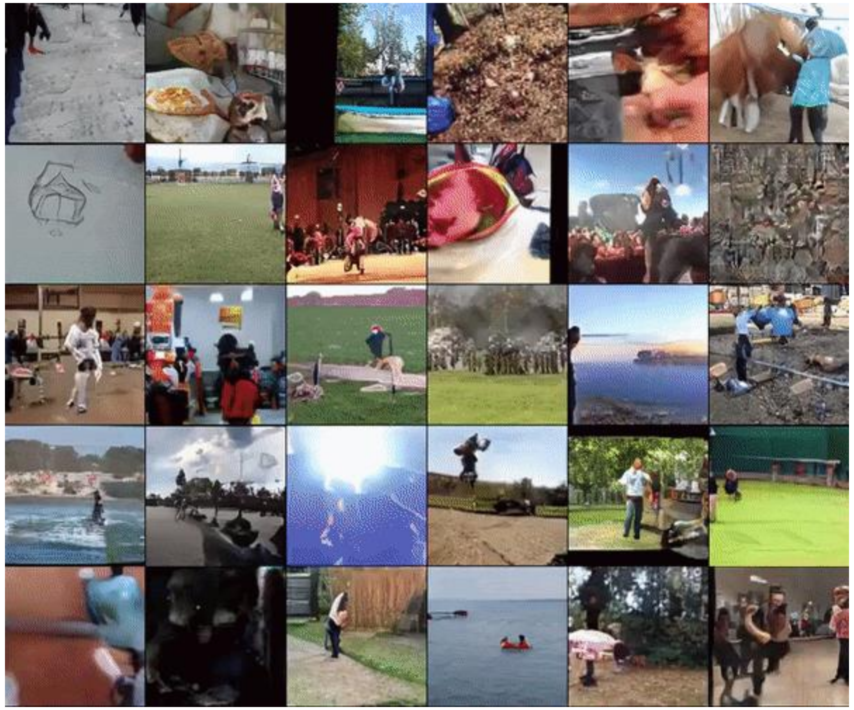
Generative Models

Learning a distribution from samples

$$x^{(1)}, x^{(2)}, \ldots, x^{(n)} \sim p_{\text{data}}(x)$$

- Traditionally, want to
  - **Compute density**: compute p(x) for some x
  - **Inference**: compute p(a|b) for some a,b
  - **Sampling**: obtain a sample from p

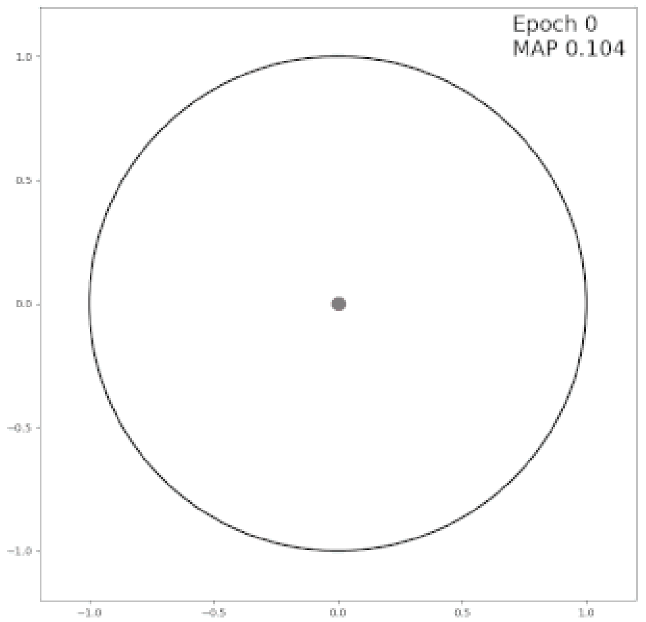
- Modern methods: may only be able to sample/conditionally sample



Embeddings & Representations

Related terminology.

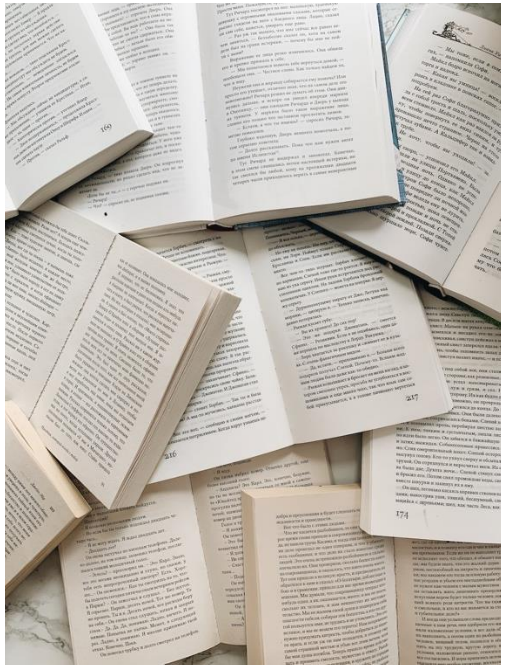
- Embeddings
  - Traditionally, goal is to take discrete objects (words, graphs, etc.) and produce vectors usable in DNNs
  - **Text**: Word2Vec **Graphs**: Hyperbolic embeddings



Embeddings & Representations

Related terminology.

- Embeddings
  - Often trained based on some custom loss (no “task”)
  - Word2Vec: word co-occurrences ⟷ embedding distances/ips

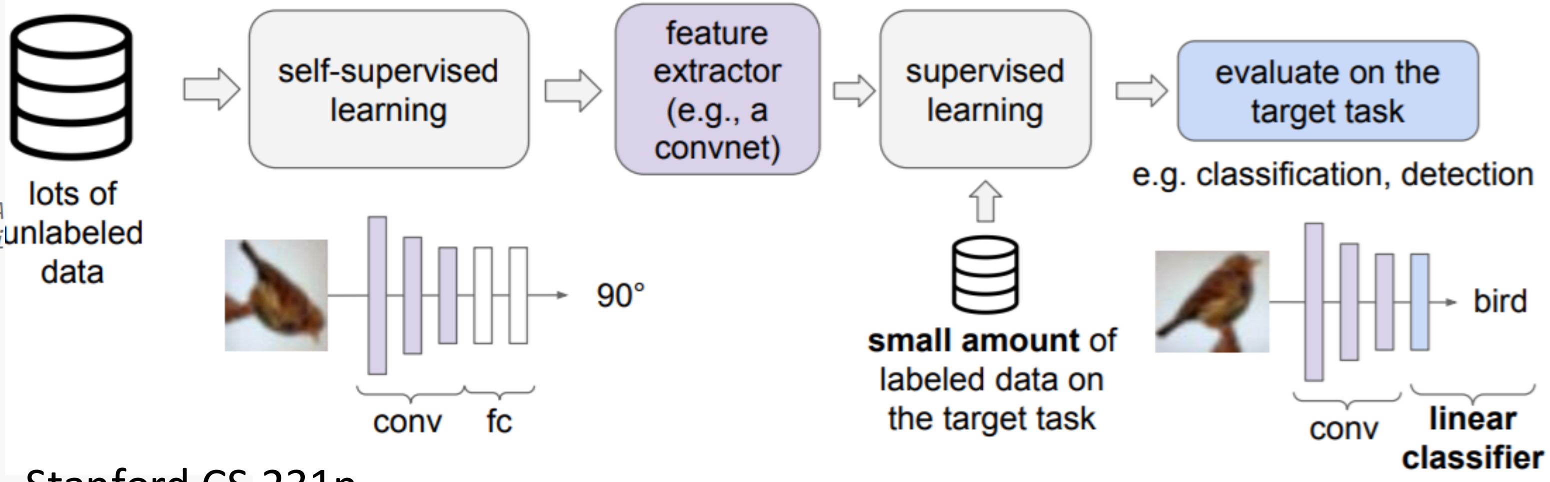


Embeddings & Representations

Related terminology.

- Representations
  - Often trained based on related task OR pretext task
  - Contain “deeper” information about each sample
  - Come from “pretrained” models

```
from torchvision.models import resnet50, ResNet50_Weights

# Old weights with accuracy 76.130%
resnet50(weights=ResNet50_Weights.IMAGENET1K_V1)

# New weights with accuracy 80.858%
resnet50(weights=ResNet50_Weights.IMAGENET1K_V2)

# Best available weights (currently alias for IMA
# Note that these weights may change across versi
resnet50(weights=ResNet50_Weights.DEFAULT)

# Strings are also supported
resnet50(weights="IMAGENET1K_V2")

# No weights - random initialization
resnet50(weights=None)
```



Stanford CS 231n