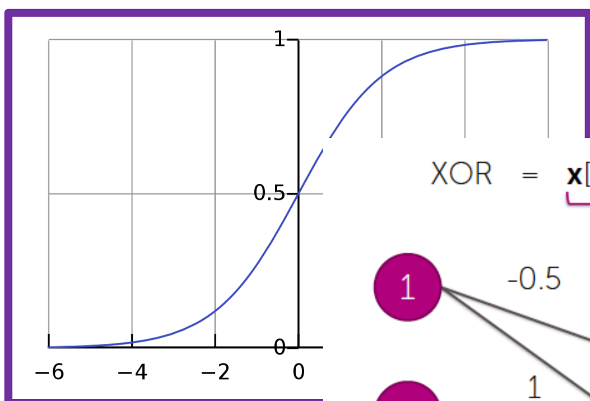
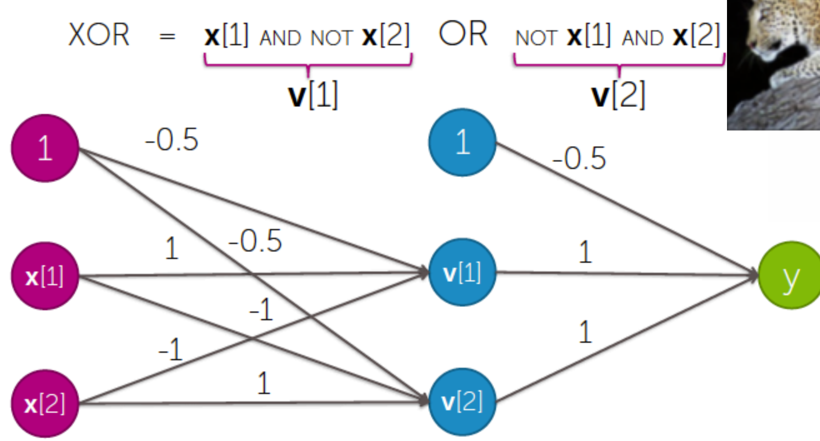
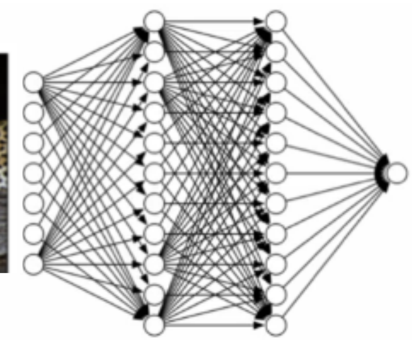
Tufts COMP 135: Introduction to Machine Learning
https://www.cs.tufts.edu/comp/135/2020f/

Neural Nets in Practice







Prof. Mike Hughes

Many slides attributable to:
Erik Sudderth (UCI), Emily Fox (UW),
Finale Doshi-Velez (Harvard)
James, Witten, Hastie, Tibshirani (ISL/ESL books)

Objectives Today: (day 13) NNs in Practice

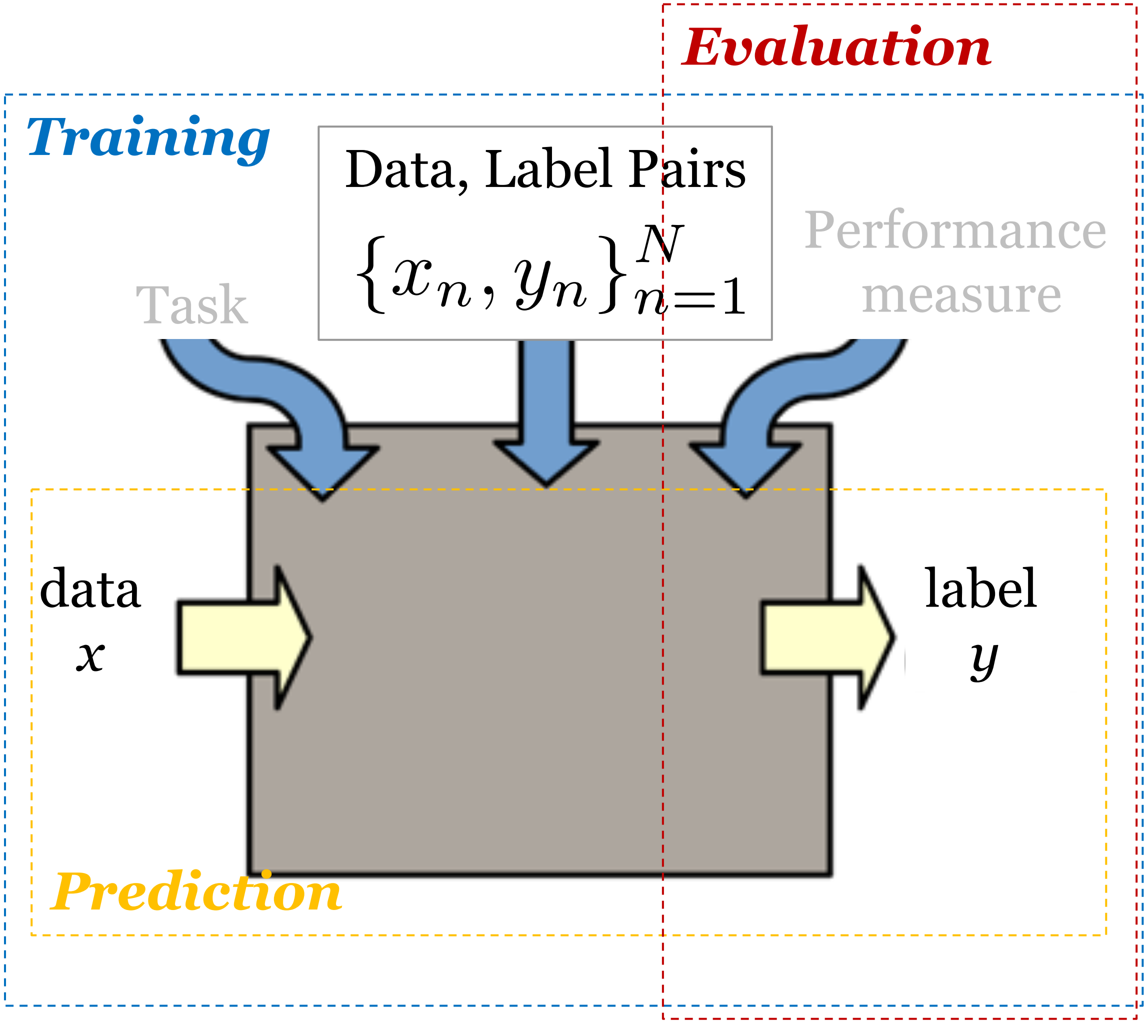
- Multi-class classification with NNs
- Pros and cons of NNs
- Avoiding overfitting with NNs
  - Hyperparameter selection
  - Data augmentation
  - Early stopping

What will we learn?

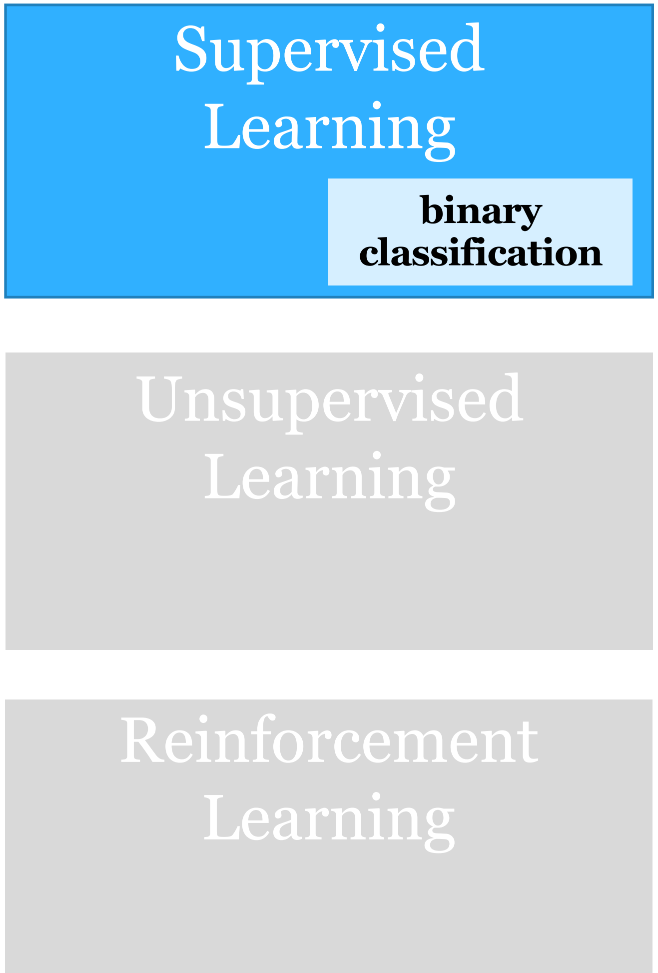
Supervised Learning

Unsupervised Learning

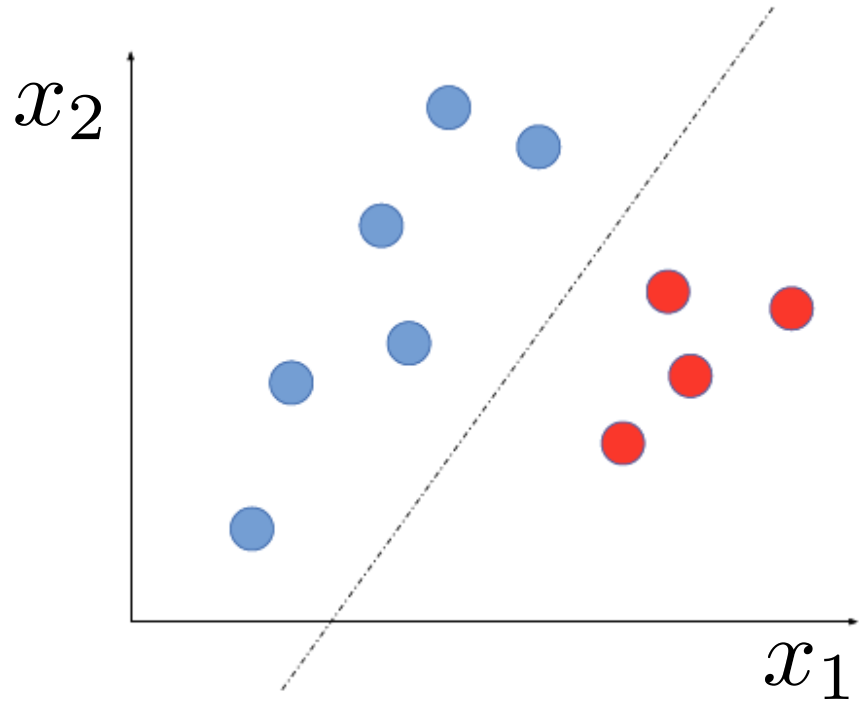
Reinforcement Learning



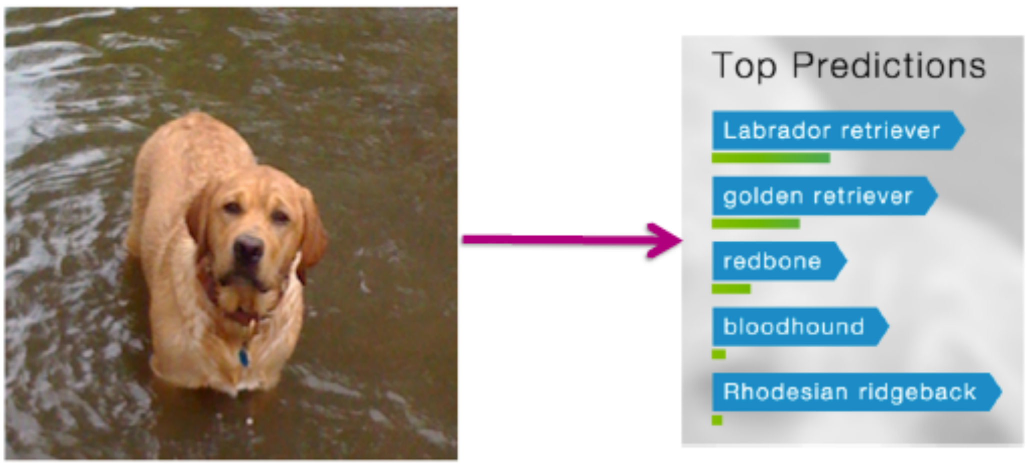
Task: Binary Classification



y is a binary variable (red or blue)



Multi-class Classification



Input: x
Image pixels

Output: y
Predicted object

How to do this?

Binary Prediction

Goal: Predict label (0 or 1) given features x

- Input: $x_i \triangleq [x_{i1}, x_{i2}, \ldots x_{if} \ldots x_{iF}]$

"features"
"covariates"
"attributes"

Entries can be real-valued, or other numeric types (e.g. integer, binary)

- Output: $y_i \in \{0, 1\}$

"responses" or "labels"

Binary label (0 or 1)

```
>>> yhat_N = model.predict(x_NF)
>>> yhat_N[:5]
[0, 0, 1, 0, 1]
```

Binary Proba. Prediction

Goal: Predict probability of label given features

- Input: $x_i \triangleq [x_{i1}, x_{i2}, \ldots x_{if} \ldots x_{iF}]$

"features"
"covariates"
"attributes"

Entries can be real-valued, or other numeric types (e.g. integer, binary)

- Output: $\hat{p}_i \triangleq p(Y_i = 1 | x_i)$

"probability"

Value between 0 and 1
e.g. 0.001, 0.513, 0.987

```
>>> yproba_N2 = model.predict_proba(x_NF)
>>> yproba1_N = model.predict_proba(x_NF)[:,1]
>>> yproba1_N[:5]
[0.143, 0.432, 0.523, 0.003, 0.994]
```

Multi-class Prediction

Goal: Predict **one of C** classes given features x

- Input: $x_i \triangleq [x_{i1}, x_{i2}, \ldots x_{if} \ldots x_{iF}]$

"features"
"covariates"
"attributes"

Entries can be real-valued, or other numeric types (e.g. integer, binary)

- Output: $y_i \in \{0, 1, 2, \ldots C - 1\}$

"responses" or "labels"

Integer label (0 or 1 or … or C-1)

```
>>> yhat_N = model.predict(x_NF)
>>> yhat_N[:6]
[0, 3, 1, 0, 0, 2]
```

Multi-class Proba. Prediction

Goal: Predict probability of **each class** given features

Input: $$x_i \triangleq [x_{i1}, x_{i2}, \ldots x_{if} \ldots x_{iF}]$$

"features"
"covariates"
"attributes"

Entries can be real-valued, or other numeric types (e.g. integer, binary)

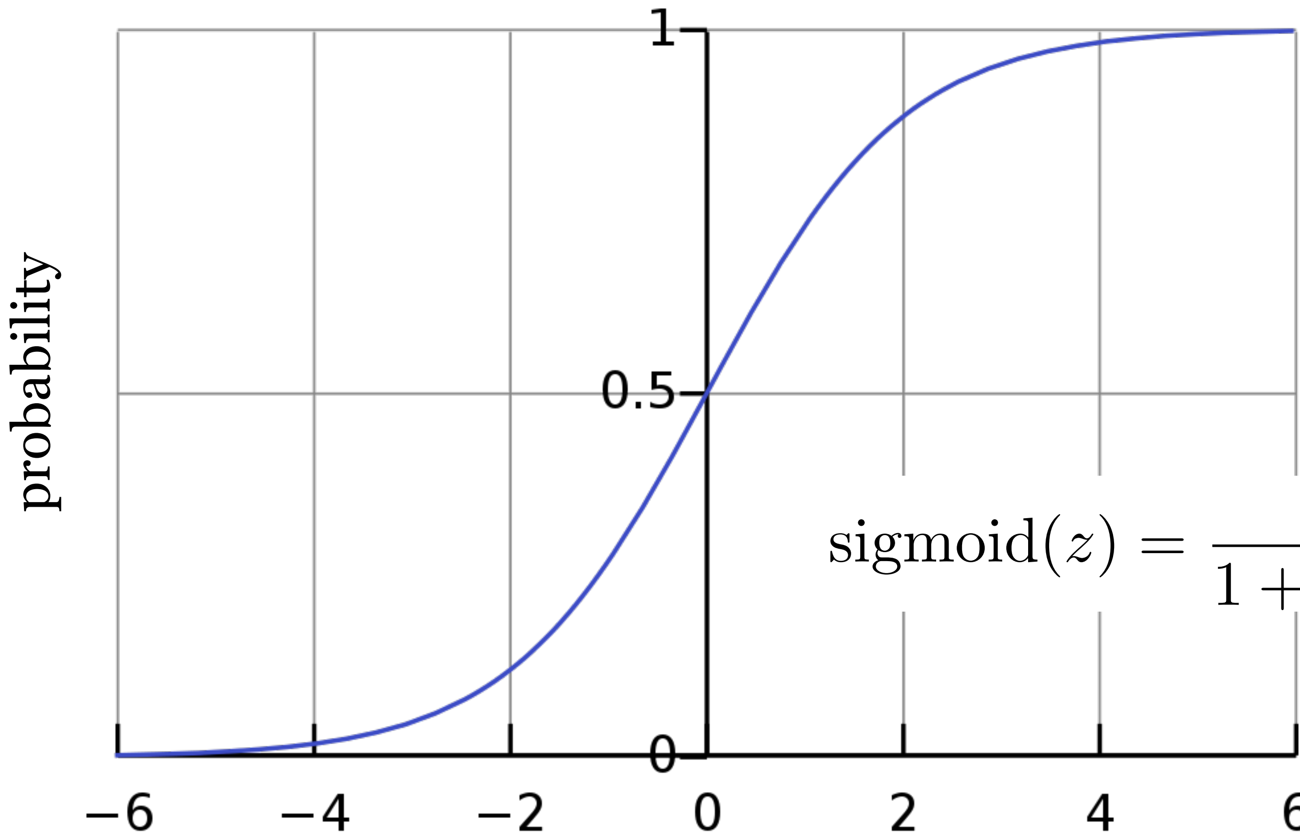
Output:

$$\hat{p}_i \triangleq [p(Y_i = 0 | x_i) \quad p(Y_i = 1 | x_i) \ldots \quad p(Y_i = C - 1 | x_i)]$$

"probability" Vector of C non-negative values, sums to one

```
>>> yproba_NC = model.predict_proba(x_NF)
>>> yproba_c_N = model.predict_proba(x_NF)[:,c]
>>> np.sum(yproba_NC, axis=1)
[1.0, 1.0, 1.0, 1.0]
```

From Real Value to Probability



$$\mathrm{sigmoid}(z) = \frac{1}{1+e^{-z}}$$

From Vector of Reals to Vector of Probabilities

$$z_i = \begin{bmatrix} z_{i1} & z_{i2} & \ldots & z_{ic} & \ldots & z_{iC} \end{bmatrix}$$

$$\hat{p}_i = \left[\frac{e^{z_{i1}}}{\sum_{c=1}^{C} e^{z_{ic}}} \quad \frac{e^{z_{i2}}}{\sum_{c=1}^{C} e^{z_{ic}}} \quad \ldots \quad \ldots \quad \frac{e^{z_{iC}}}{\sum_{c=1}^{C} e^{z_{ic}}} \right]$$

called the "softmax" function

Representing multi-class labels

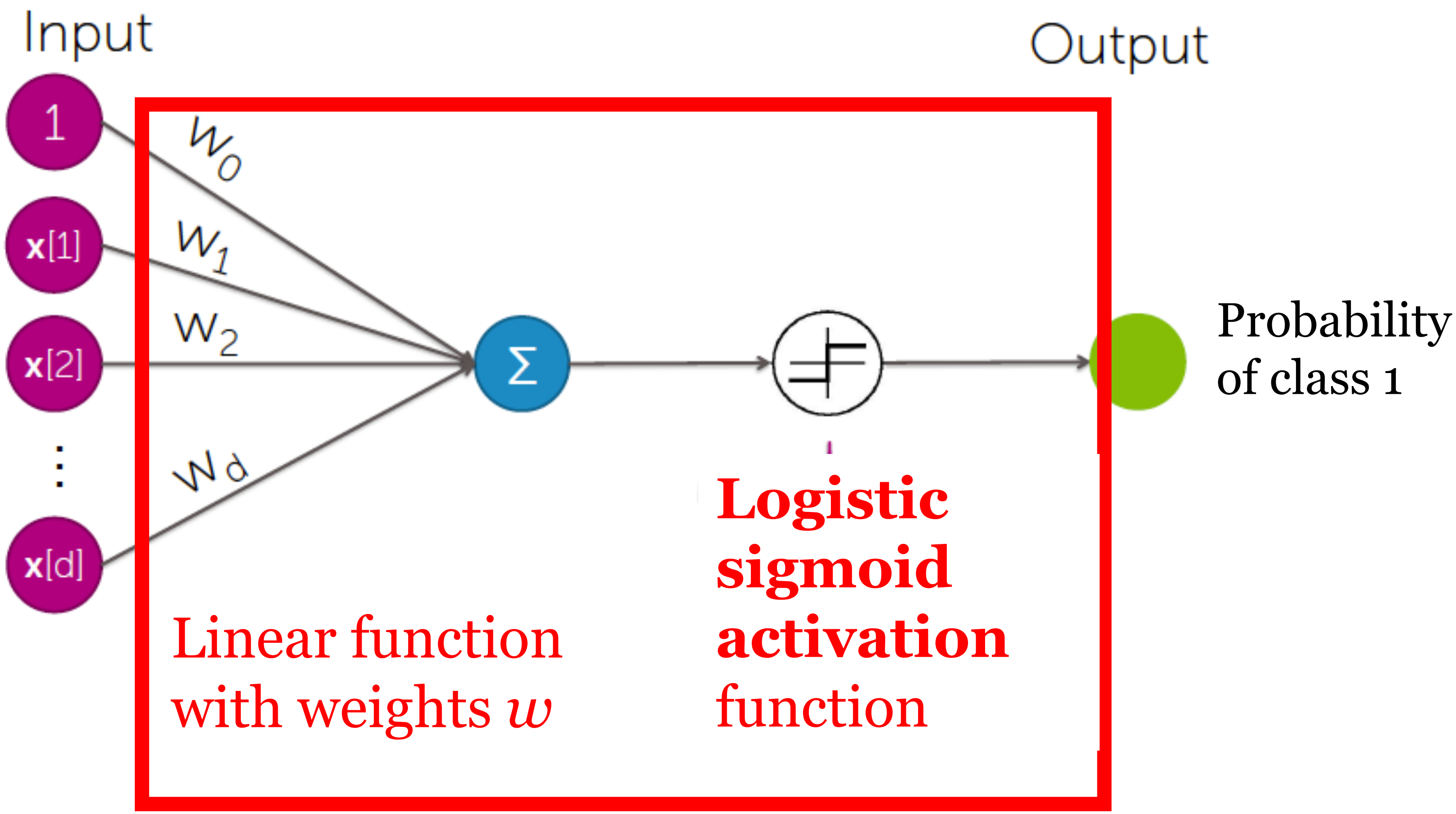
$$y_n \in \{0, 1, 2, \dots C-1\}$$

Encode as length-C ***one hot binary*** vector

$$\bar{y}_n = \begin{bmatrix} \bar{y}_{n1} & \bar{y}_{n2} & \dots & \bar{y}_{nc} & \dots & \bar{y}_{nC} \end{bmatrix}$$

```
Examples (assume C=4 labels)

    class 0:   [1 0 0 0]
    class 1:   [0 1 0 0]
    class 2:   [0 0 1 0]
    class 3:   [0 0 0 1]
```

“Neuron” for Binary Prediction

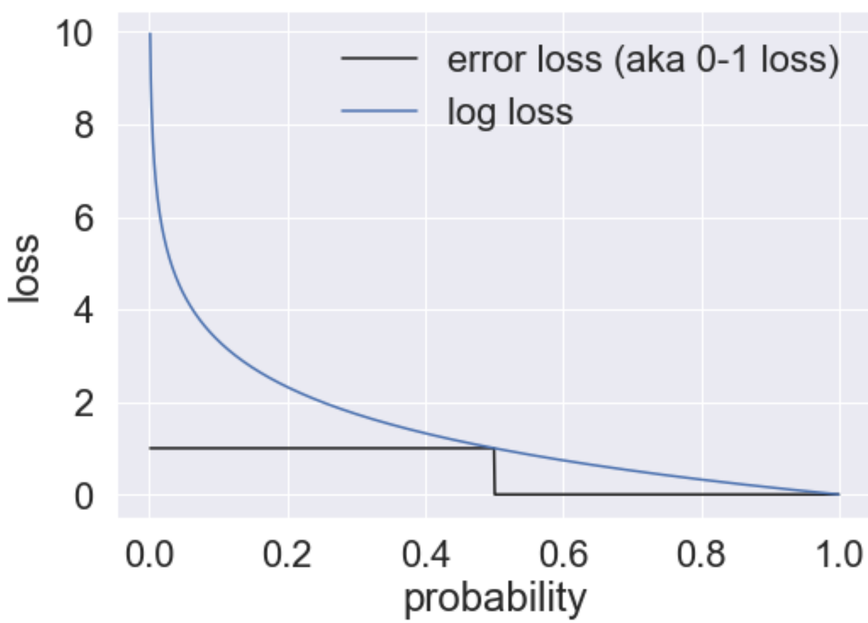


Credit: Emily Fox (UW)

Recall: Binary log loss

$$\text{error}(y, \hat{y}) = \begin{cases} 1 & \text{if } y \neq \hat{y} \\ 0 & \text{if } y = \hat{y} \end{cases}$$

$$\text{log\_loss}(y, \hat{p}) = -y \log \hat{p} - (1 - y) \log(1 - \hat{p})$$



Plot assumes:

- True label is 1

- Threshold is 0.5

- Log base 2

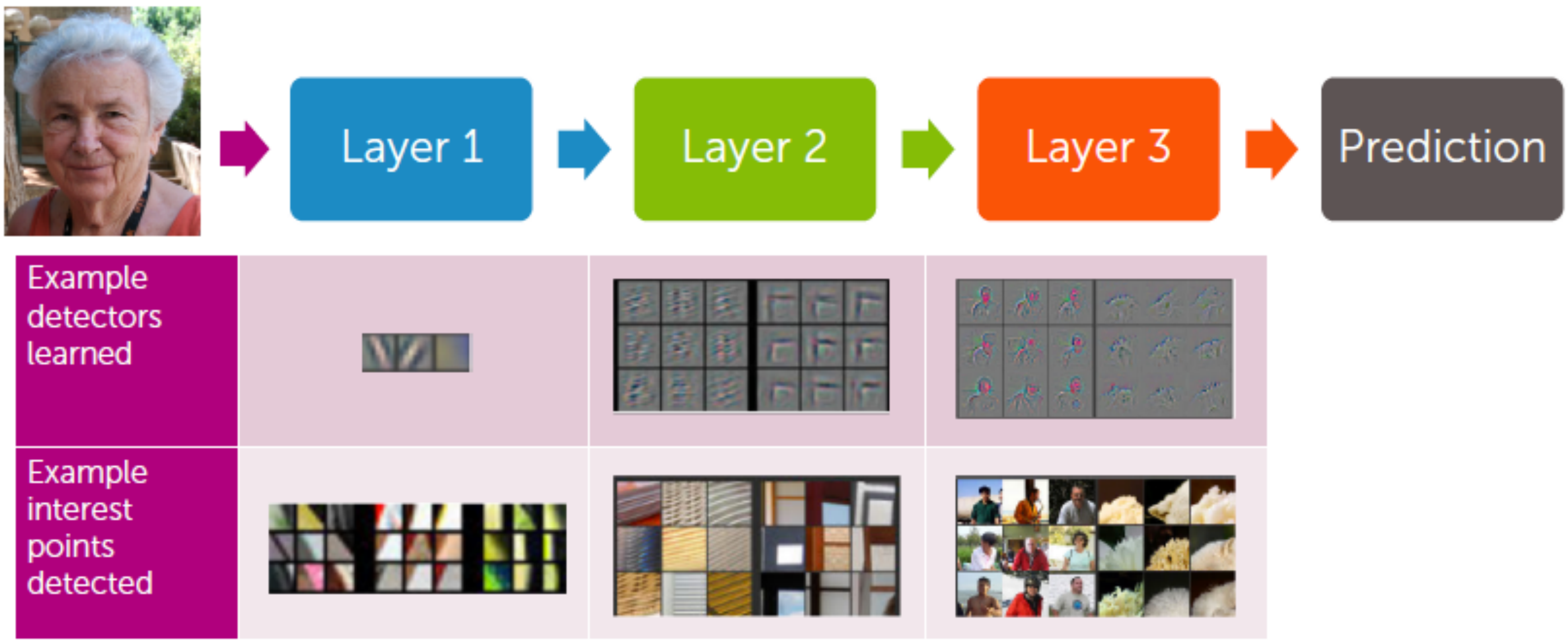
Multi-class log loss

Input: two vectors of length C
Output: scalar value (strictly non-negative)

$$\text{log\_loss}(\bar{y}_n, \hat{p}_n) = -\sum_{c=1}^{C} \bar{y}_{nc} \log \hat{p}_{nc}$$

Justifications carry over from the binary case:
- Interpret as upper bound on the error rate
- Interpret as cross entropy of multi-class discrete random variable
- Interpret as log likelihood of multi-class discrete random variable

Each Layer Extracts "Higher Level" Features



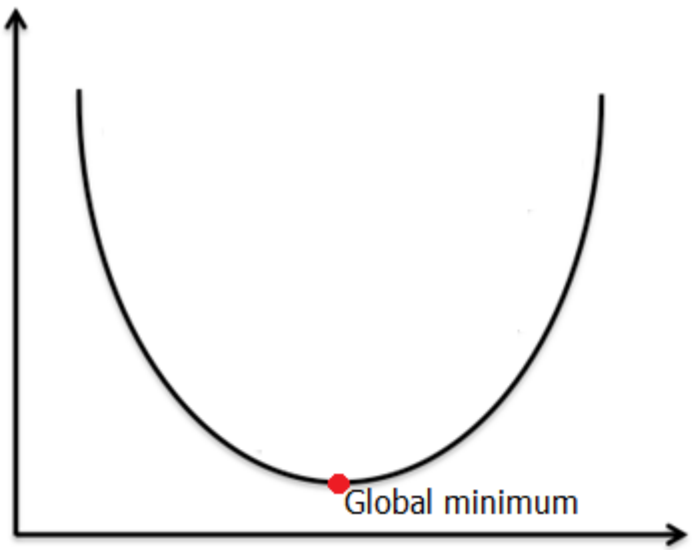
Deep Neural Nets

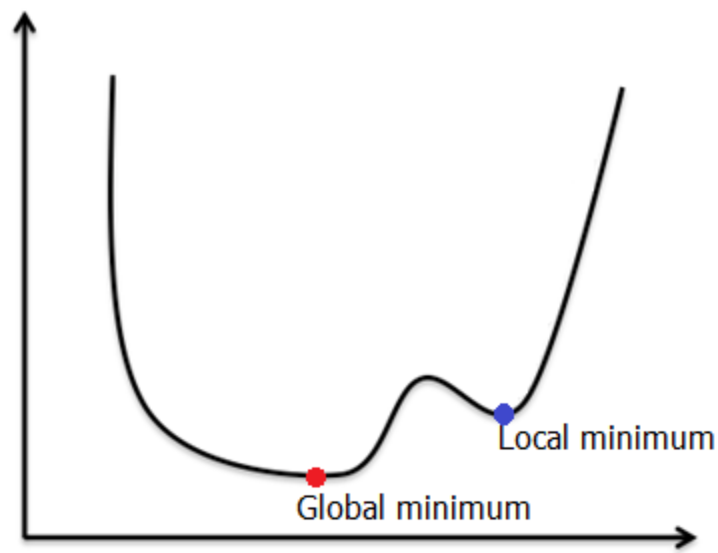
PROs

- Flexible models
- State-of-the-art success in many applications
  - Object recognition
  - Speech recognition
  - Language models
- Open-source software

CONs?

Two kinds of optimization problem





Convex

Only one global minimum
If GD converges, solution is best possible

Non-Convex

One or more local minimum
GD solution might be much worse than global minimum

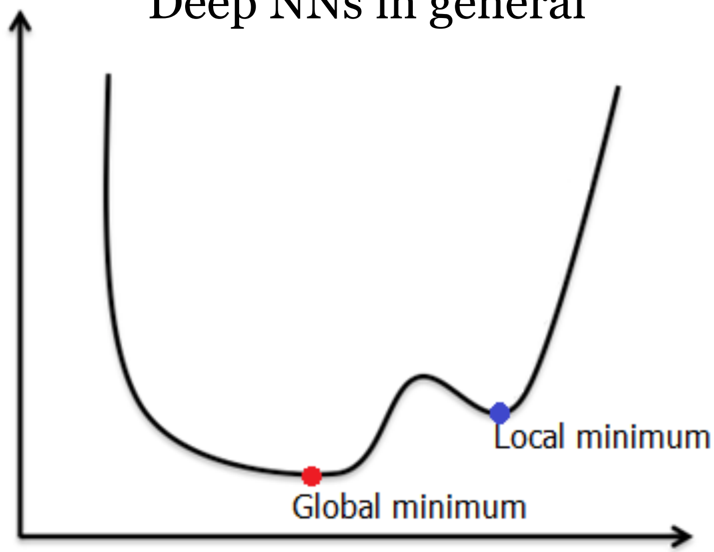
Deep Neural Nets: Optimization is **not** convex

Linear regression
Logistic regression

Convex

Only one global minimum
If GD converges, solution is best possible

MLPs with 1+ hidden layers
Deep NNs in general



Non-Convex

One or more local minimum
GD solution might be much worse than global minimum

Deep Neural Nets

PROs

- Flexible models
- State-of-the-art success in many applications
  - Object recognition
  - Speech recognition
  - Language models
- Open-source software

CONs

- Require lots of data
- Each run of SGD can take hours/days
- Optimization not easy
  - Will it converge?
  - Is local minimum good enough?
- Hard to extrapolate
- Many hyperparameters
- Will it overfit?

Many hyperparameters for a Deep Neural Network (MLP)

- Num. layers
- Num. units / layer
- Activation function
- L2 penalty strength

Control model complexity

- Learning rate
- Batch size

Optimization quality/speed

Guidelines: complexity params

- Num. units / layer
  - Start with similar to num. features
  - Add more (log spaced) until serious overfitting
- Num. layers
  - Start with 1
  - Add more (+1 at a time) until serious overfitting
- L2 penalty strength scalar
  - Try range of values (log spaced)
- Activation function
  - ReLU for most problems is reasonable

Grid Search

1) Choose candidate values of each hyperparameter

Step size/learning rate $\{0.1, 0.01, 10^{-3}, 10^{-4}, 10^{-5}\}$

Number of hidden units $\{50, 100, 200, 500, 1000, 2000\}$

2) For each combination, assess its **heldout score**

- We need to choose in advance:
  - Performance metric (e.g. AUROC, log loss, TPR at PPV > 0.98, etc.)
    - What is most important for your task?
  - Source of heldout data
    - Fixed validation set : *Faster, simpler*
    - Cross validation with K folds : *Less noise, better use of all available data*

3) Select the one single combination with **best** score

Grid Search

1) Choose candidate values of each hyperparameter

Step size/learning rate $\{0.1, 0.01, 10^{-3}, 10^{-4}, 10^{-5}\}$

Number of hidden units $\{50, 100, 200, 500, 1000, 2000\}$

2) For each combination, assess its **heldout score**

- We need to choose in advance:
  - Performance metric (e.g. AUROC, log loss, TPR at PPV > 0.98, etc.)
    - What is most important for your task?
  - Source of heldout data
    - Fixed validation set : *Faster, simpler*
    - Cross validation with K folds : *Less noise, better use of all available data*

3) Select the one single combination with **best** score

Each trial can be parallelized. Can do for numeric or discrete variables.
But, number of combinations to try can quickly grow infeasible

Random Search

1) Choose candidate **distributions** of each hyperparameter

Usually, for convenience, assume each independent

number of hidden units was drawn geometrically[3] from 18 to 1024.

learning rate ε_0 drawn geometrically from 0.001 to 10.0

2) For each of T **samples**, assess **heldout score**

- We need to choose in advance:
  - Performance metric (e.g. AUROC, log loss, TPR at PPV > 0.98, etc.)
    - What is most important for your task?
  - Source of heldout data
    - Fixed validation set : *Faster, simpler*
    - Cross validation with K folds : *Less noise, better use of all available data*

3) Select the one single combination with **best** score

Random Search

1) Choose candidate **distributions** of each hyperparameter

Usually, for convenience, assume each independent

number of hidden units was drawn geometrically[3] from 18 to 1024.

learning rate ε_0 drawn geometrically from 0.001 to 10.0

2) For each of T **samples**, assess **heldout score**

- We need to choose in advance:
  - Performance metric (e.g. AUROC, log loss, TPR at PPV > 0.98, etc.)
    - What is most important for your task?
  - Source of heldout data
    - Fixed validation set : *Faster, simpler*
    - Cross validation with K folds : *Less noise, better use of all available data*

3) Select the one single combination with **best** score

Each trial can be parallelized. Best for numeric values.
Benefits in **coverage** over grid search.

Random Search covers more of the parameter space

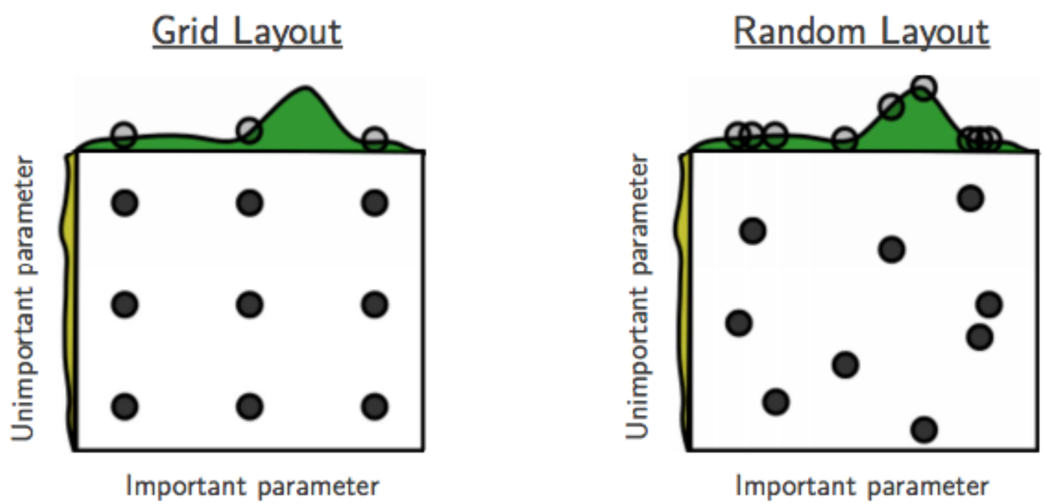
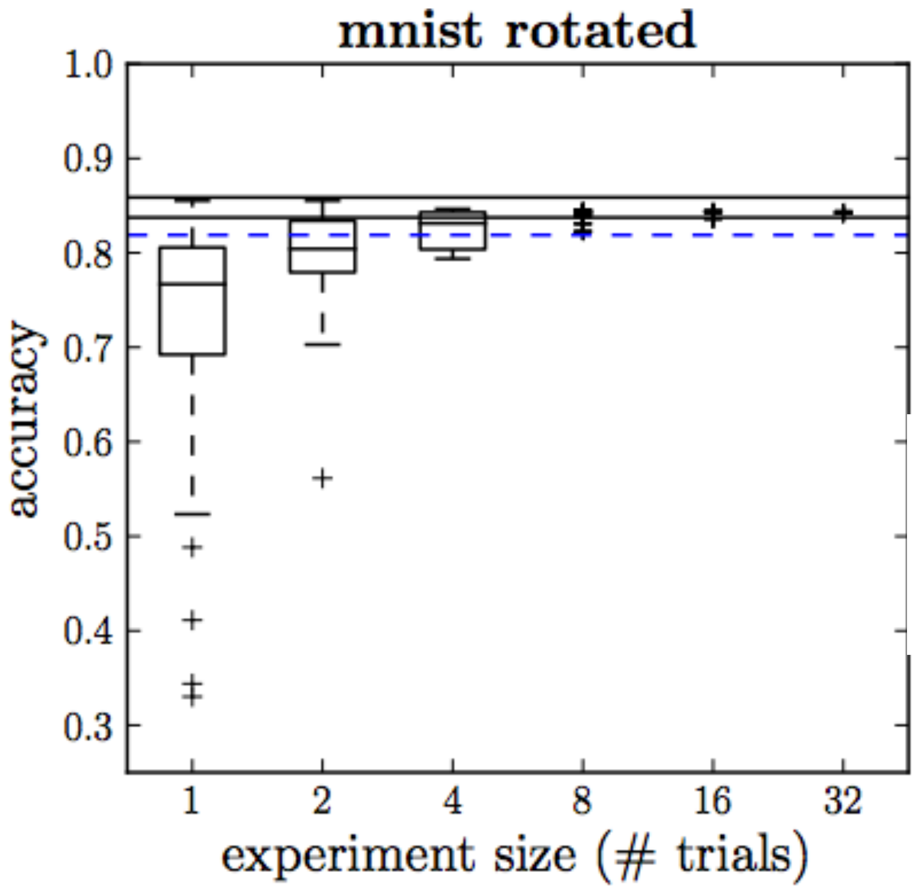


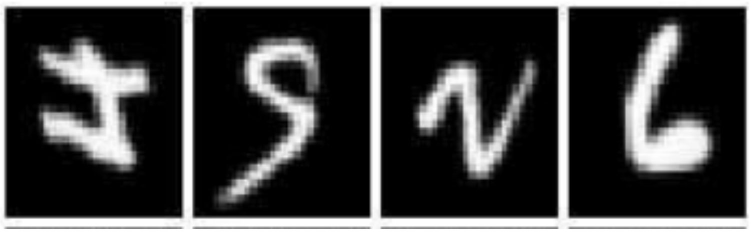
Figure 1: Grid and random search of nine trials for optimizing a function $f(x,y) = g(x) + h(y) \approx g(x)$ with low effective dimensionality. Above each square $g(x)$ is shown in green, and left of each square $h(y)$ is shown in yellow. With grid search, nine trials only test $g(x)$ in three distinct places. With random search, all nine trials explore distinct values of g. This failure of grid search is the rule rather than the exception in high dimensional hyper-parameter optimization.

Credit: Bergstra & Bengio JMLR 2012

8 random trials beats 100 grid search trials on MNIST digits



Grid search over 100 configs



Credit: Bergstra & Bengio JMLR 2012

Hyperopt Toolbox

https://www.youtube.com/watch?v=Mp1xnPfE4PY

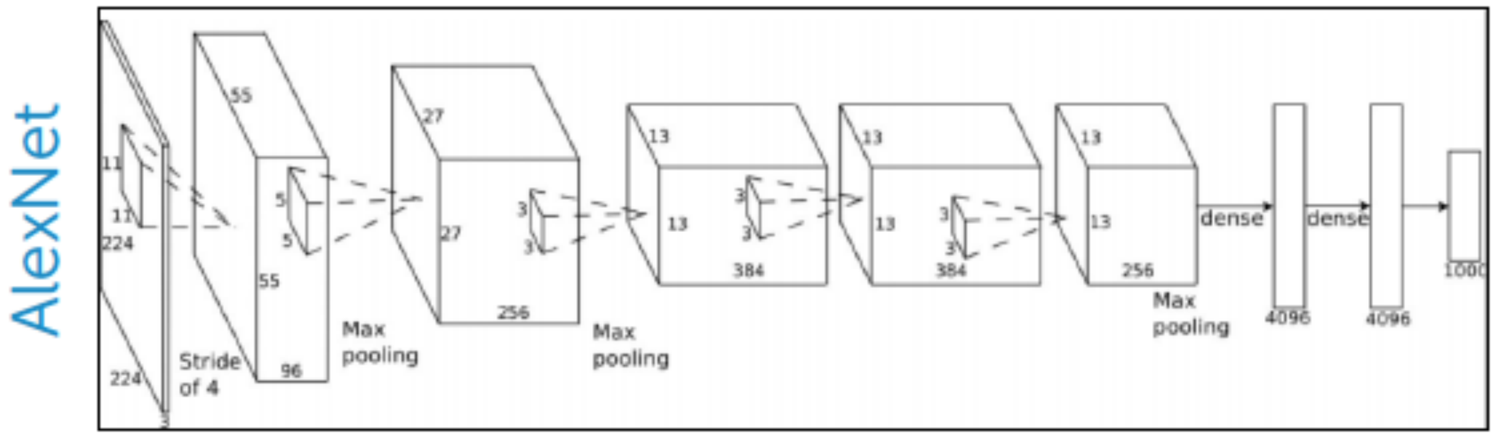
https://github.com/hyperopt/hyperopt/wiki/FMin

```python
from hyperopt import fmin, tpe, rand, hp

def loss(x):
    return x**2

best_rand_search = fmin(fn=loss,
    space=hp.uniform('x', -10, 10),
    algo=rand.suggest,
    max_evals=100)

best_tpe_search = fmin(fn=loss,
    space=hp.uniform('x', -10, 10),
    algo=tpe.suggest,
    max_evals=100)
```

2012 ImageNet Challenge Winner

ImageNet challenge
1000 categories, 1.2 million images in training set

8 layers, 60M parameters [Krizhevsky et al. '12]

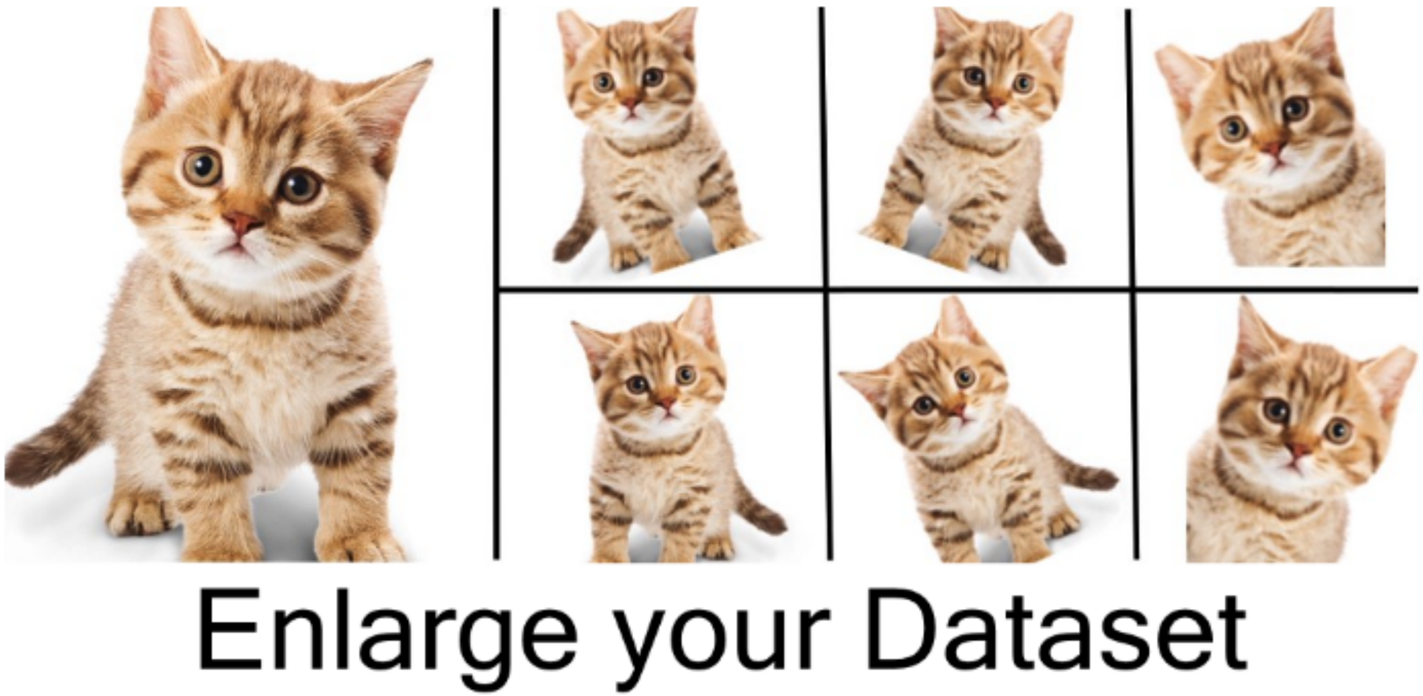


How to learn 60 million parameters from 1 million examples?

NN Tricks to avoid overfitting

- Gather more data
  - *Data augmentation*
- Modify optimization
  - *Early stopping*

Data Augmentation: Gather more (artificial) data



Credit: Bharath Raj (medium.com post)

Data Augmentation

Data Augmentation: Increase effective size of training dataset by applying perturbations to existing features x to create new (x', y) pairs

Choose perturbations which do not change label.

Images
- Flip left-to-right
- Slight rotations or crops
- Recolor or brighten

Text
- Add slight misspellings
- Replace word with similar word

This scheme approximately captures an important property of natural images, namely, that object identity is invariant to changes in the intensity and color of the illumination. This scheme reduces the top-1 error rate by over 1%.

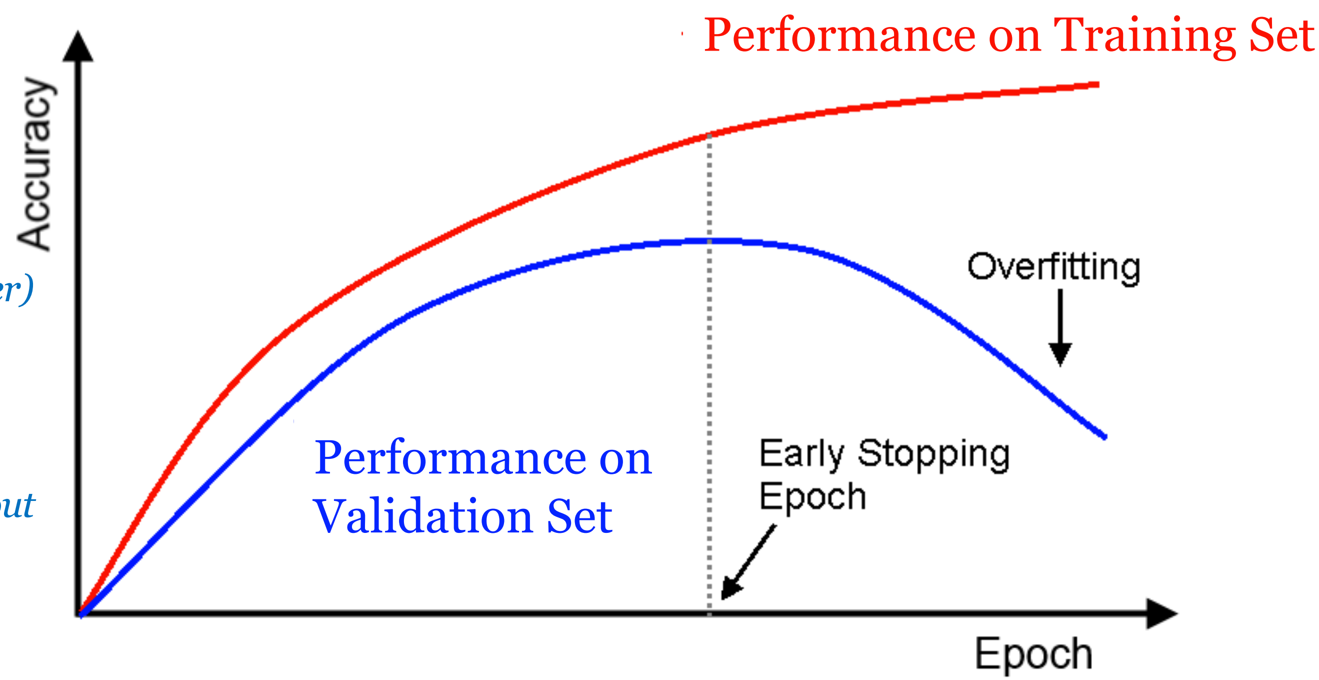
from AlexNet paper (Krizhevsky et al. NIPS 2012)

Early Stopping

Performance Metric

(assume higher is better)

Could be accuracy,
area under ROC,
Recall, precision,
whatever you care about



Big idea: stop training after your heldout validation set stops improving

- Avoid overfitting
- Save time / compute resources

Credit: https://deeplearning4j.org/docs/latest/deeplearning4j-nn-early-stopping

Objectives Today: (day 13) NNs in Practice

- Multi-class classification with NNs
- Pros and cons of NNs
- Avoiding overfitting with NNs
  - Hyperparameter selection
  - Data augmentation
  - Early stopping