Training ≠ Testing

- **Generalization performance**
  (the empirical distribution lies)
- **Covariate shift**
  (the covariate distribution lies)
- **Logistic regression**
  (tools to fix shift)
- **Covariate shift correction**
- **Label shift**
  (the label distribution lies)
- **Nonstationary Environments**

$$p_{\text{emp}}(x, y) \neq p(x, y)$$

$$p(x) \neq q(x)$$

$$\log(1 + \exp(-yf(x)))$$

$$\frac{1}{2}\left(p(x)\delta(1, y) + q(x)\delta(-1, y)\right)$$

$$p(y) \neq q(y)$$

aws

Generalization performance

Generalization performance








Generalization performance








Generalization performance









aws

Generalization performance





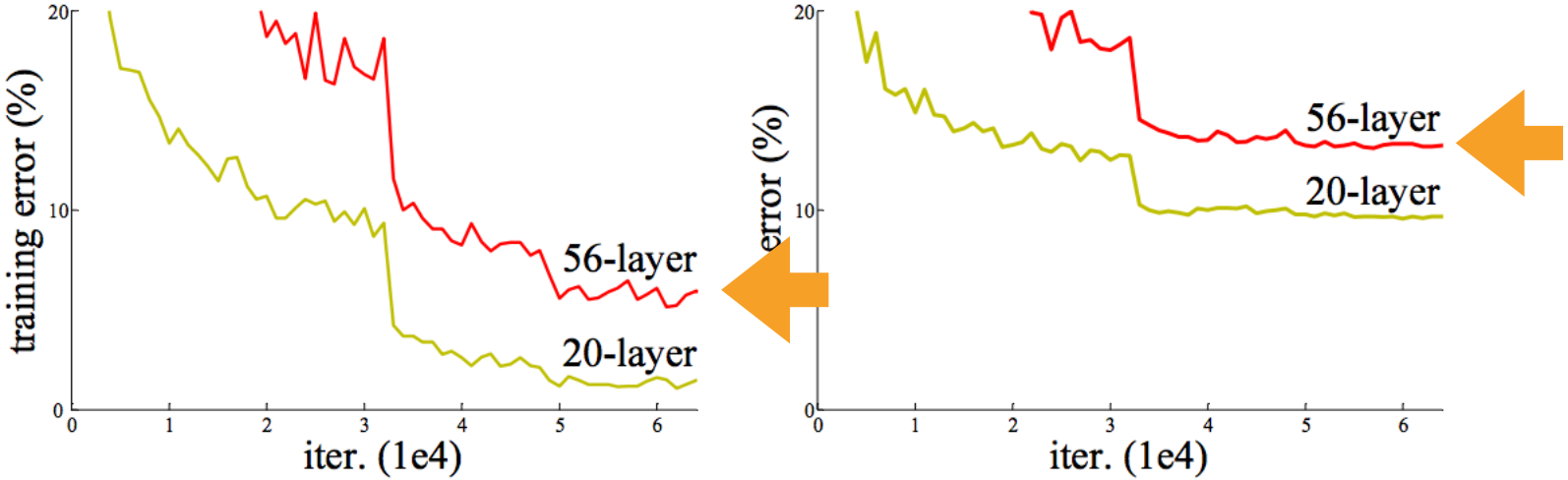




Generalization performance

Only cats and dogs?

- Images, too (e.g. He et al., 2015, ResNet paper)



- Alexa
  (‘Please turn off the coffee machine’ vs. ‘coffee machine off’)

Why?

- Data Distribution p(x,y)
- Dataset drawn from p(x,y)
- Training minimizes empirical risk (plus regularization)

$$\underset{w}{\text{minimize}} \frac{1}{m} \sum_{I=1}^{m} l(f(x_i, w), y_i)$$

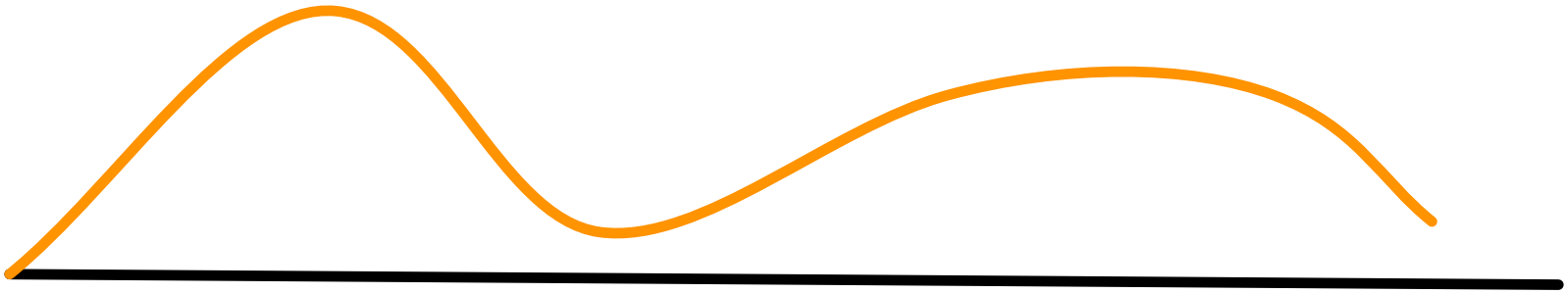
- At test time expected risk matters
  (all the other data we could have seen)

$$\mathbf{E}_{(x,y)\sim p} \left[l(f(x, w), y) \right]$$

aws

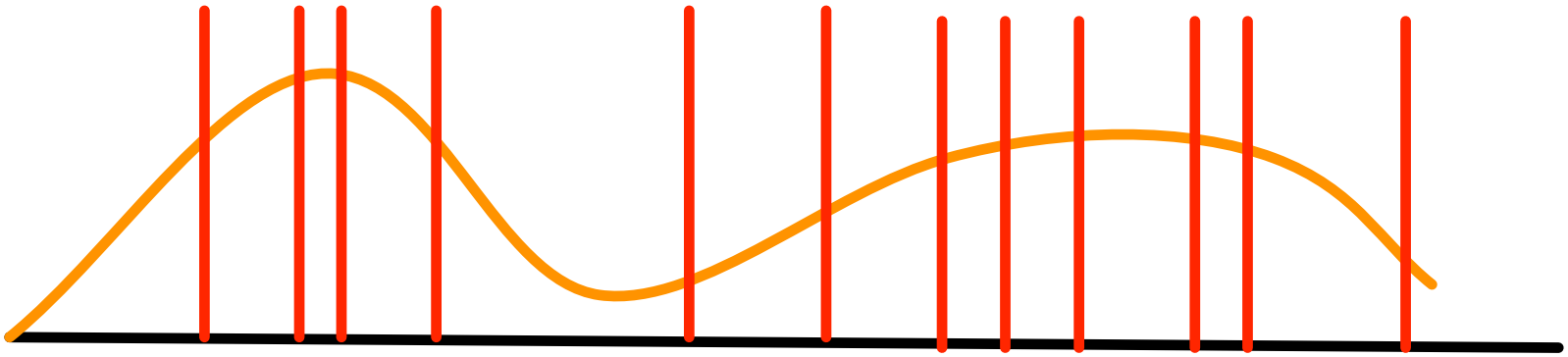
Why

Data Distribution



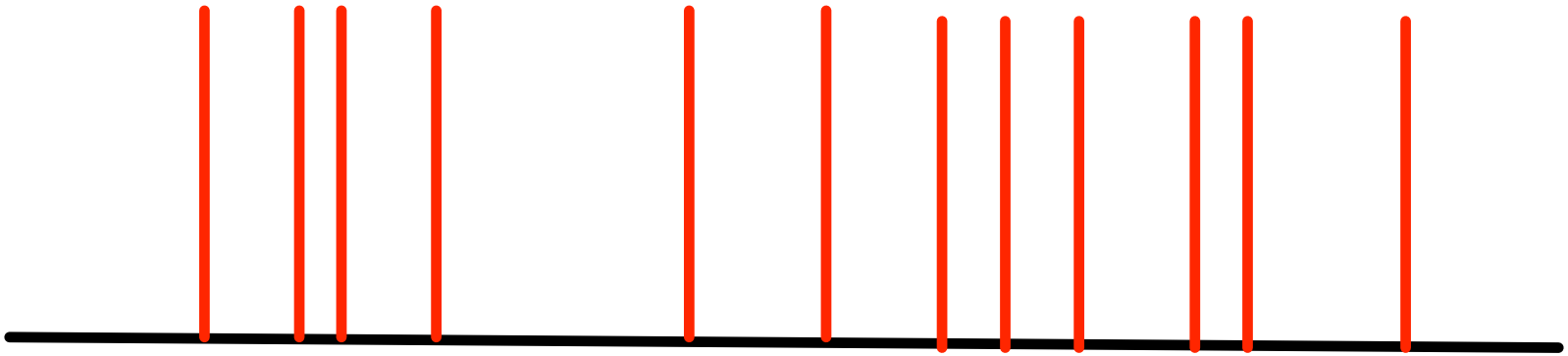
Why

Data Distribution with Empirical Sample



Why

Empirical Sample



Fixing it

- **Validation set**
  (hold out **separate** data that is not used for training)
- **Chernoff bound**

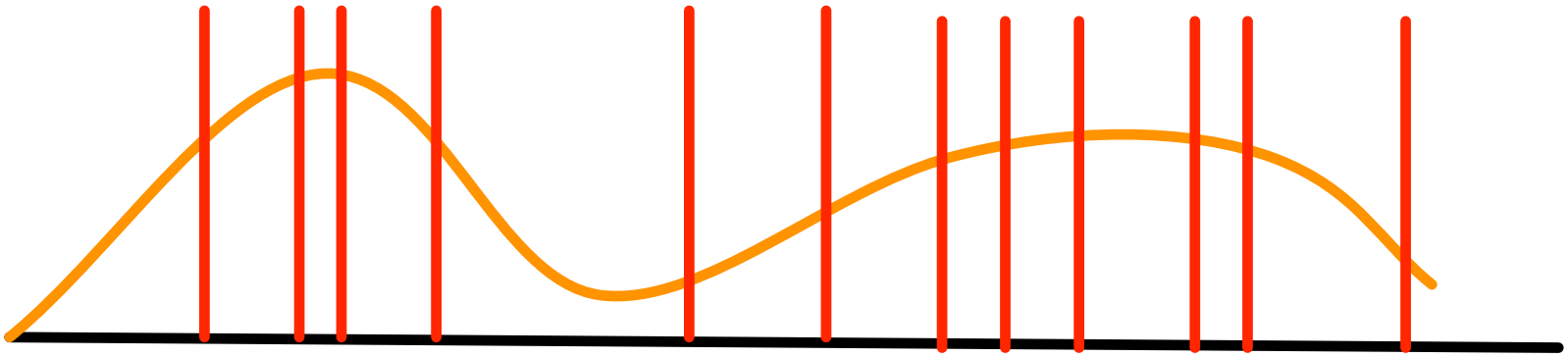
$$\Pr\left\{\frac{1}{m}\sum_{I=1}^{m} l(f(x_i), y_i)) - \mathbf{E}\left[l(f(x), y)\right] > \epsilon\right\} \leq \exp\left(-2m\epsilon^2\right)$$

- **Why does it work?**
  - Validation set was never used for training (often violated)
  - Loss bounded within [0,1] (otherwise rescale)

aws

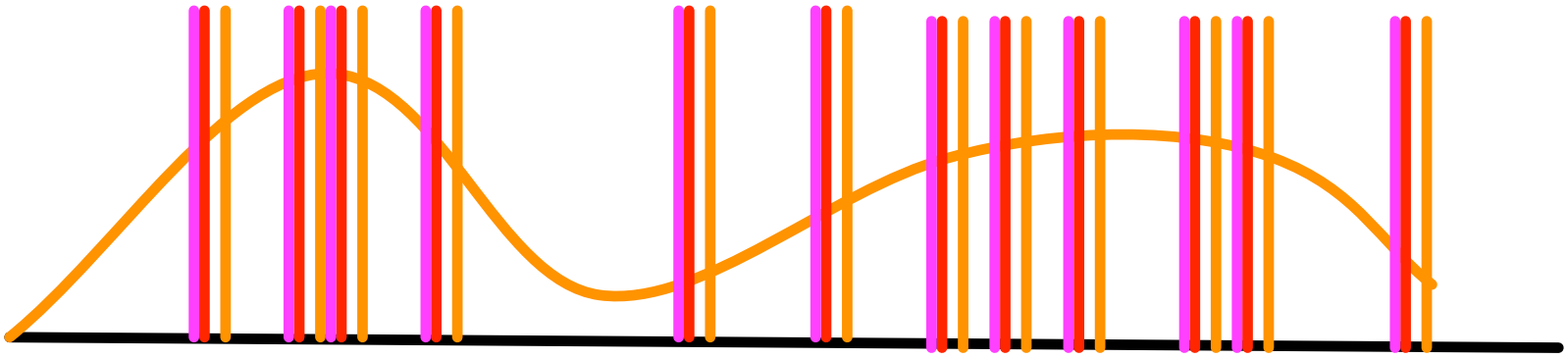
Fixing it

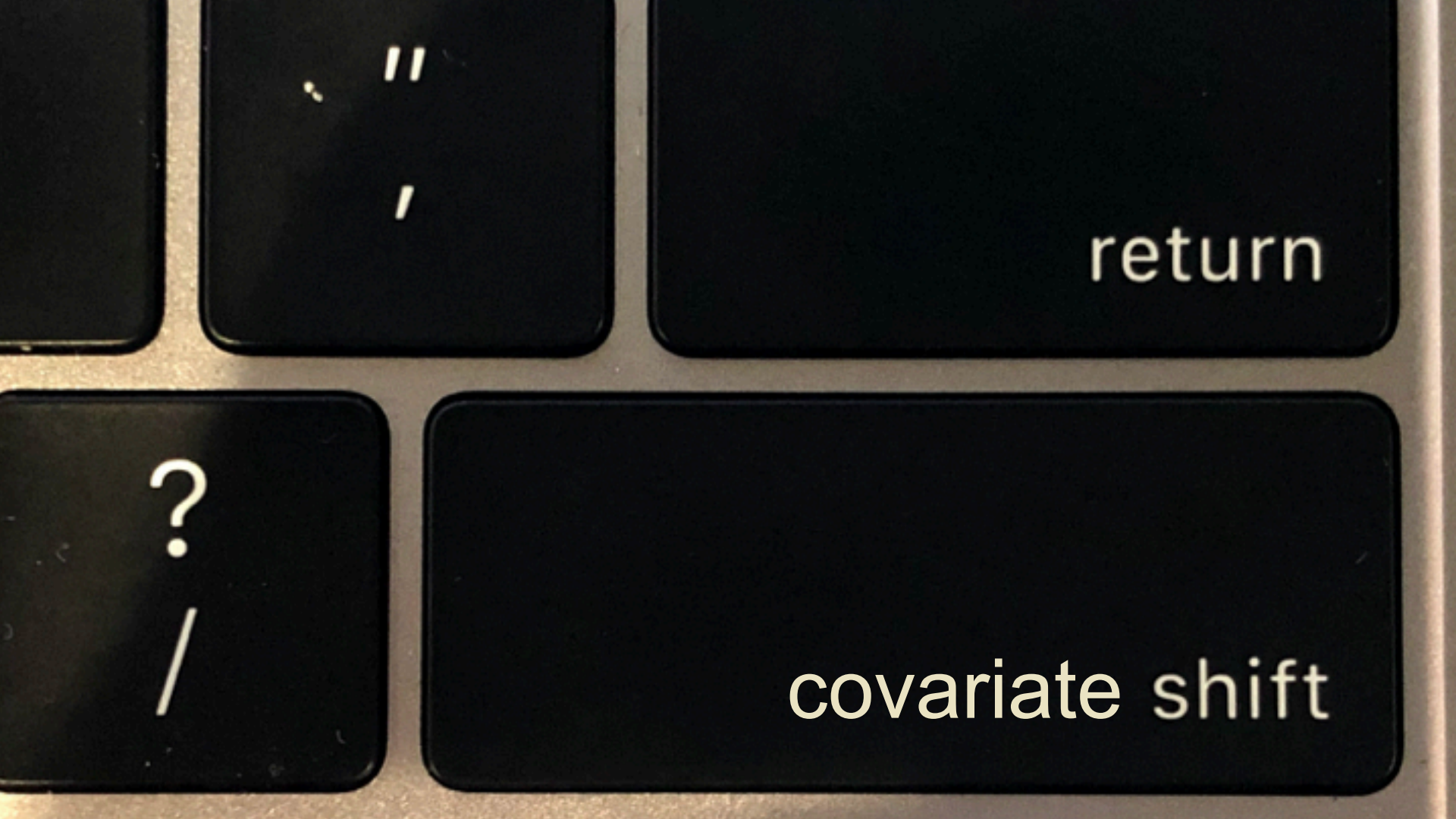
Data Distribution with Empirical Sample



Fixing it

- Input noise (more on this later)
- Dropout (noise within the layers)
- Smoothing the function f (e.g. weight decay)




"
'
return
?
/
covariate shift

Training set








Training set

At test time

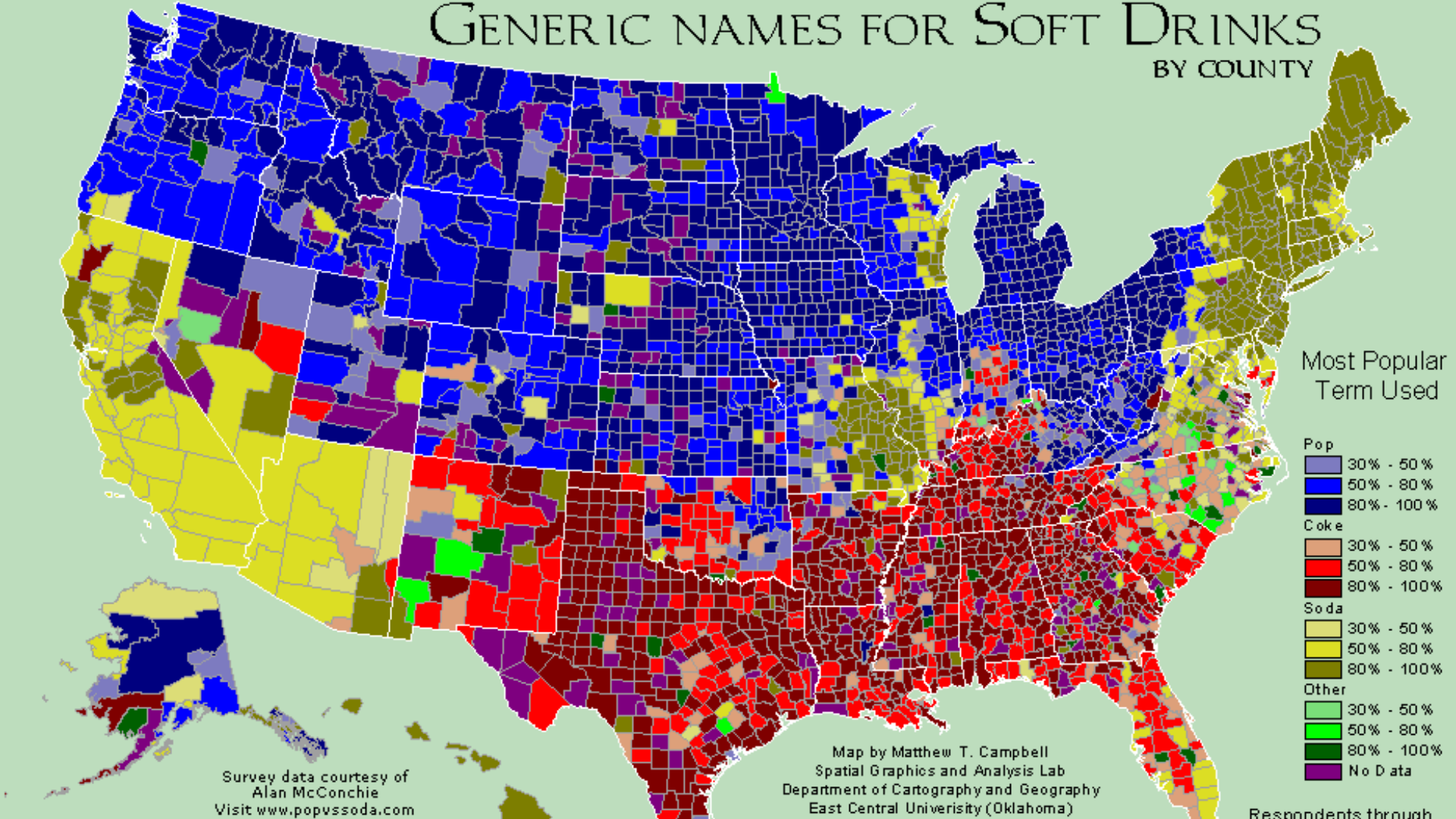
Why would anyone do this?

Covariate Shift

- **Web search**
  - Training - page relevance data for the US market
  - Testing - recommend pages for Canada (UK, Australia)
- **Speech recognition**
  - Training - West coast accent
  - Testing - Southern drawl, Texan, non-native speaker
- **Language**
  - Training - ‘James, bring me a **soda**’
  - Testing - ‘John, bring me a ‘**pop**’ (or coke, etc.)

aws

Generic names for Soft Drinks
by county



Covariate Shift

- **Medical**
  - Training - University students + old men with prostate cancer
  - Testing - Potentially sick old men
- **Reinforcement Learning**
  - Training - Data gathered with current policy
  - Testing - Environment reacting to updated policy
- **Databases**
  - Training - DB tuned to 2017 usage pattern
  - Testing - DB deployed on AWS in 2018

aws

What is happening? $q(x, y) = q(x)p(y|x)$

- Training Risk

Training data

$$\underset{w}{\text{minimize}} \int dx\, p(x) \int dy\, p(y|x) l(f(x, w), y)$$

$$\text{or rather} \quad \underset{w}{\text{minimize}} \frac{1}{m} \sum_{I=1}^{m} l(f(x_i, w), y_i)$$

- Test Risk is different

Test data

$$\int dx\, q(x) \int dy\, p(y|x) l(f(x, w), y)$$

aws

Fixing it (covariate shift correction)

- Basic algebra

$$\int dx\, q(x) f(x) = \int dx\, p(x) \underbrace{\frac{q(x)}{p(x)}}_{\alpha(x)} f(x) = \int dx\, p(x) \alpha(x) f(x)$$

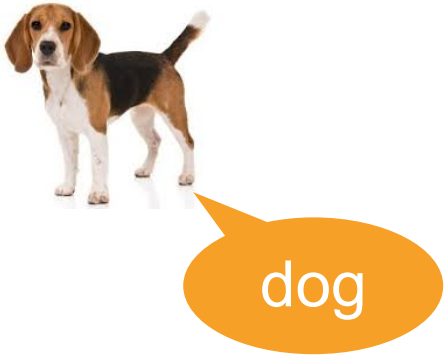
- Need to find density ratio, but we don't have either one.
- Estimating p and q directly is really hard and requires specialized tools. Can we recycle classifiers?

aws

Fairness and Bias
(covariate shift in action & news)

aws

Training set



Test set

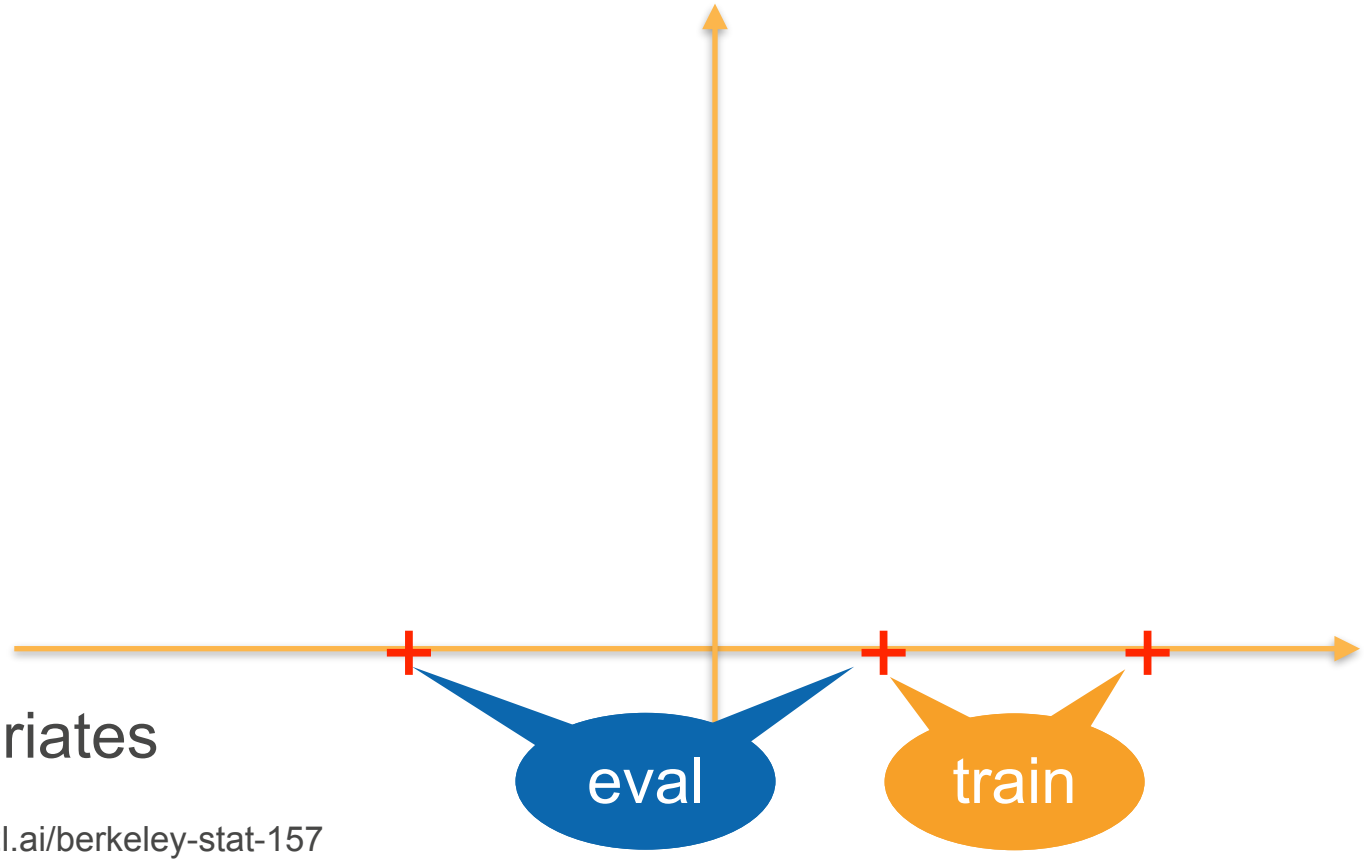
The classifier might perform a lot worse during test time

aws

Why?

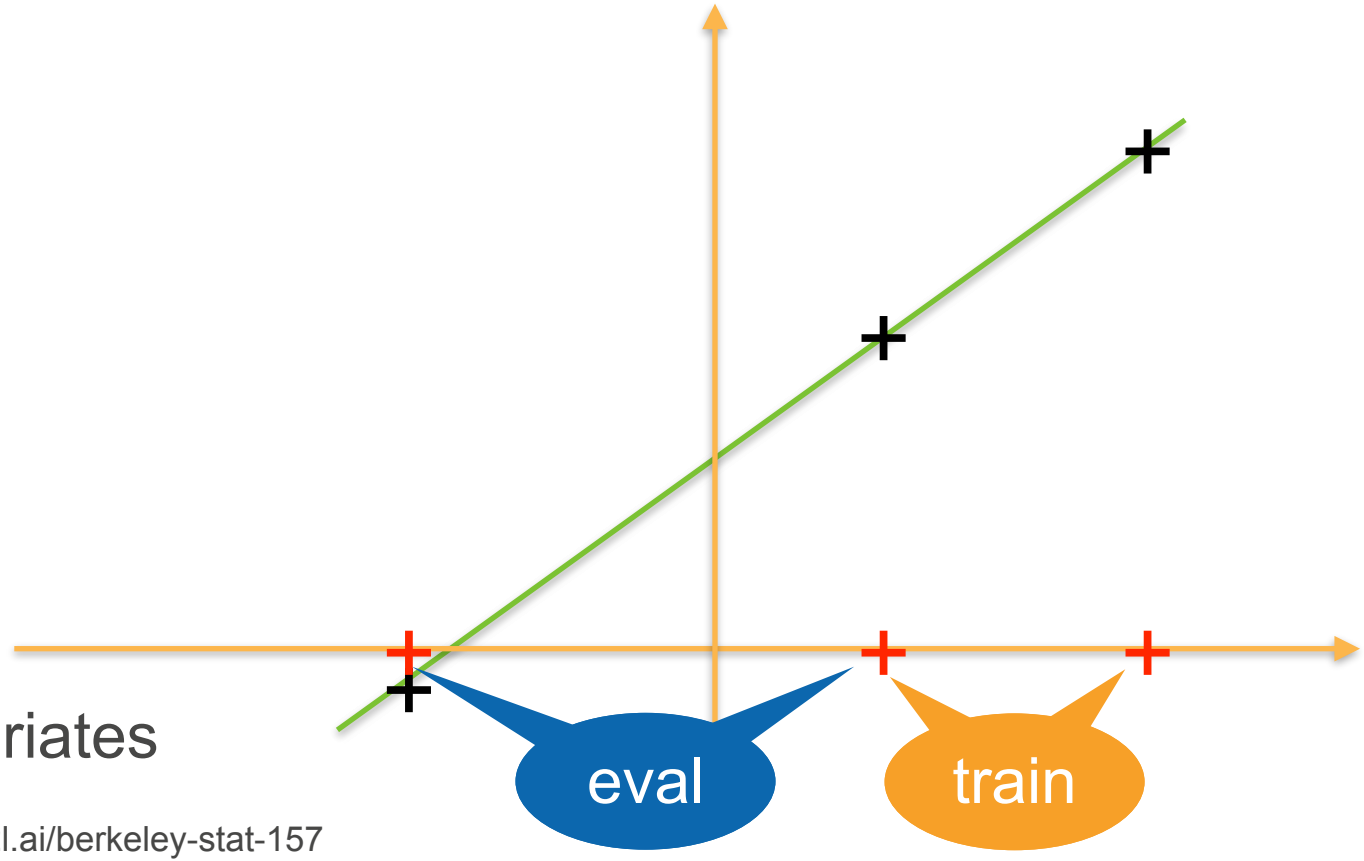
aws

Simple regression problem



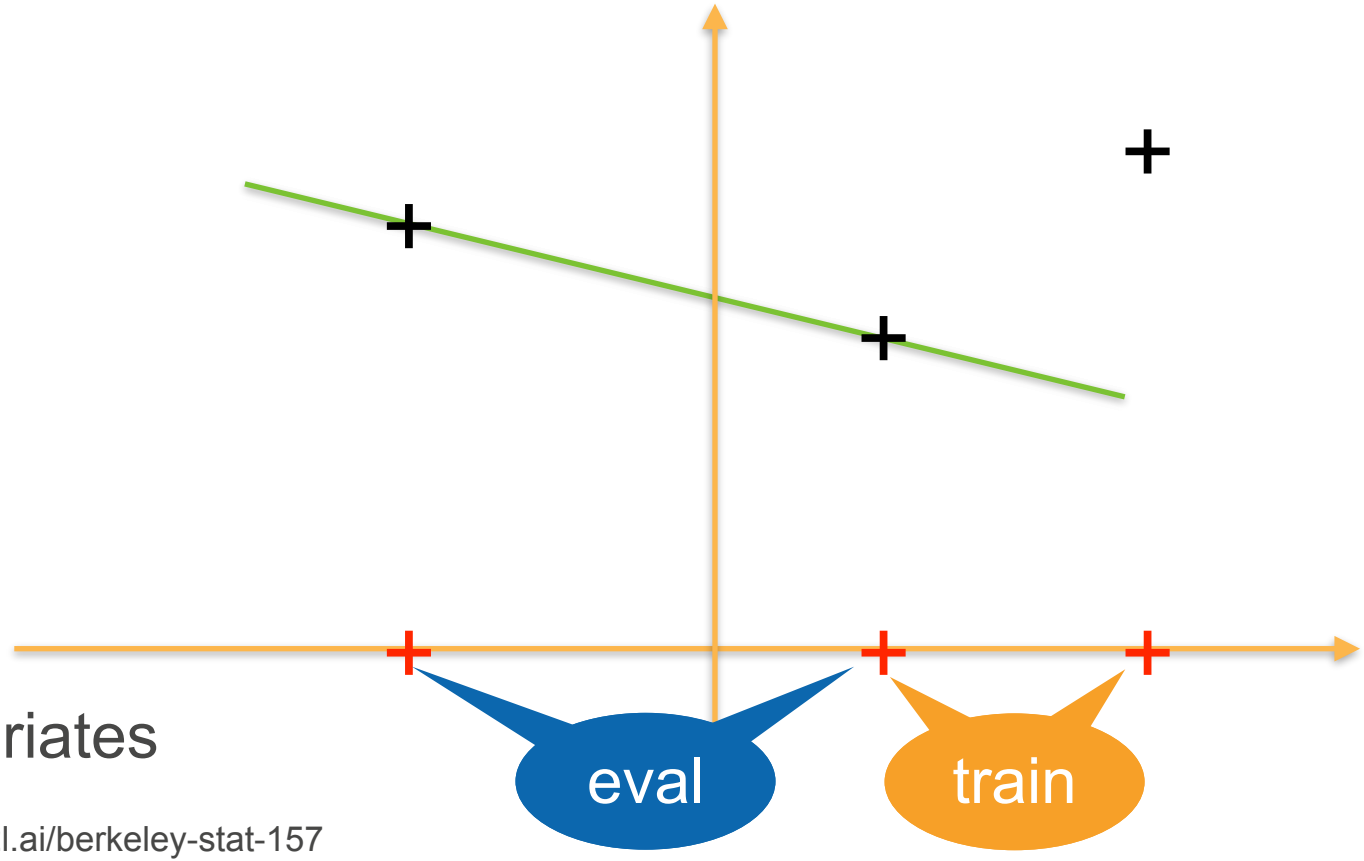
aws

Simple regression problem



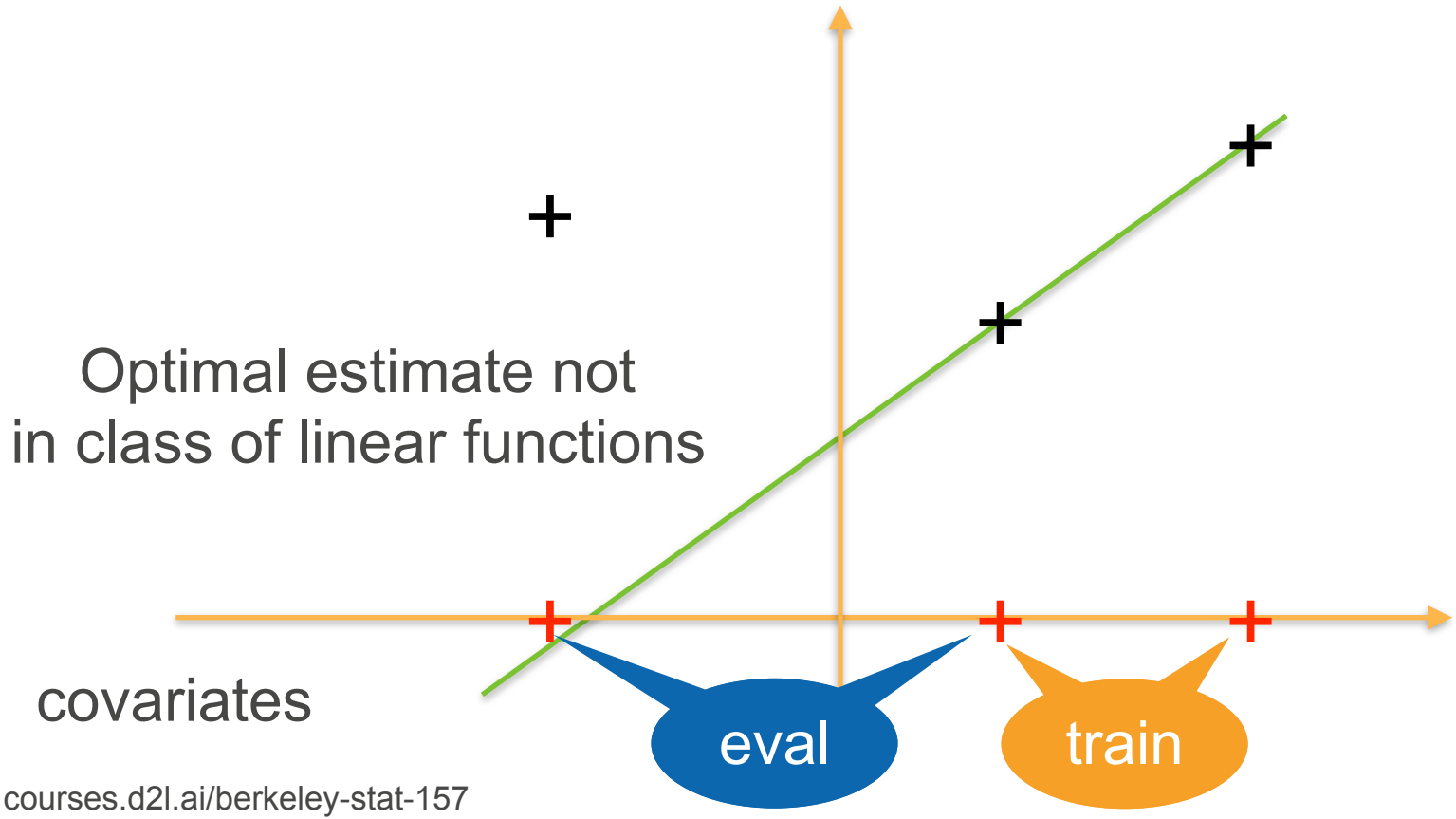
aws

Simple regression problem

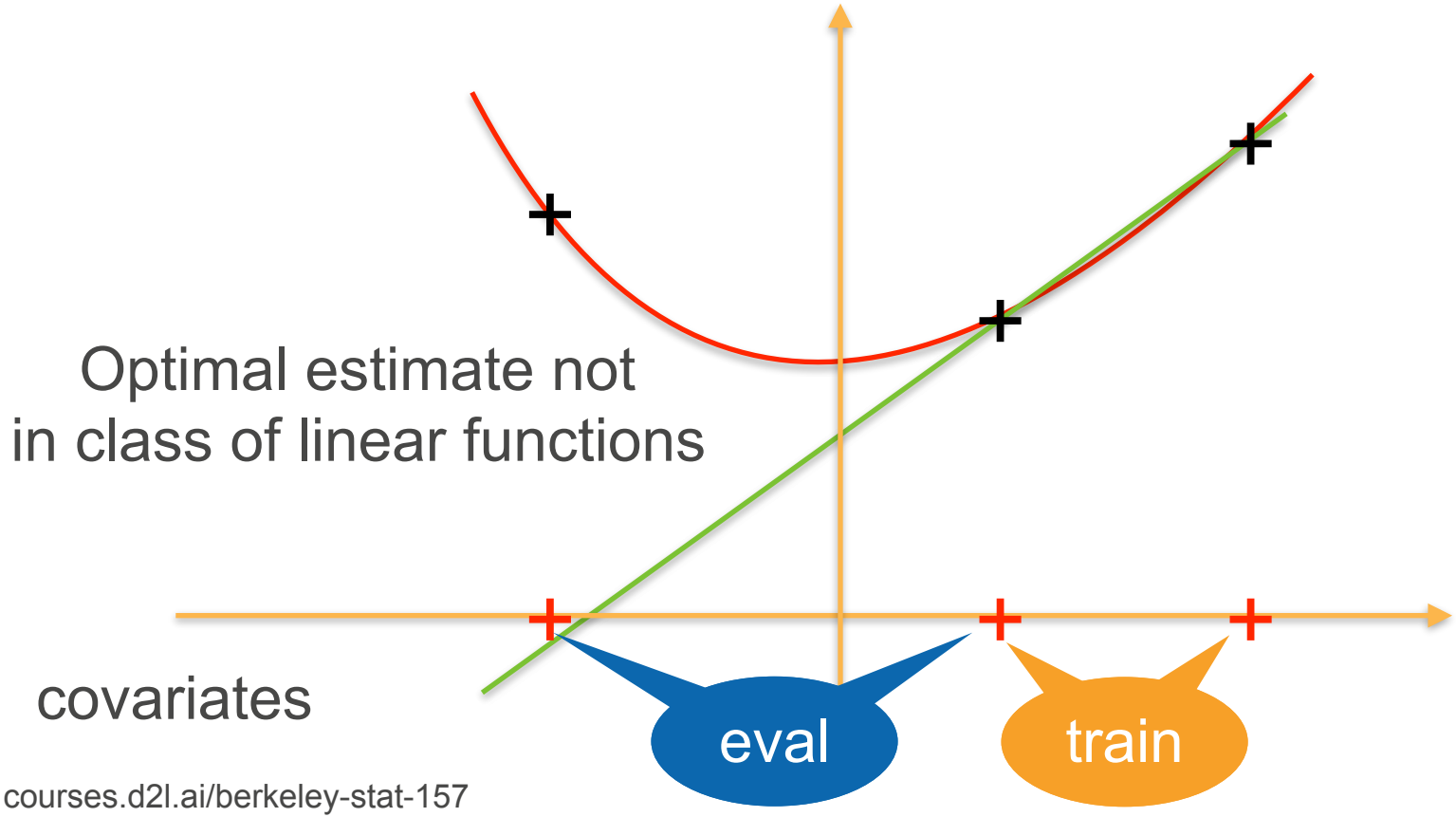


aws

Simple regression problem



Simple regression problem



aws

Training error may be misleading (e.g. faces)

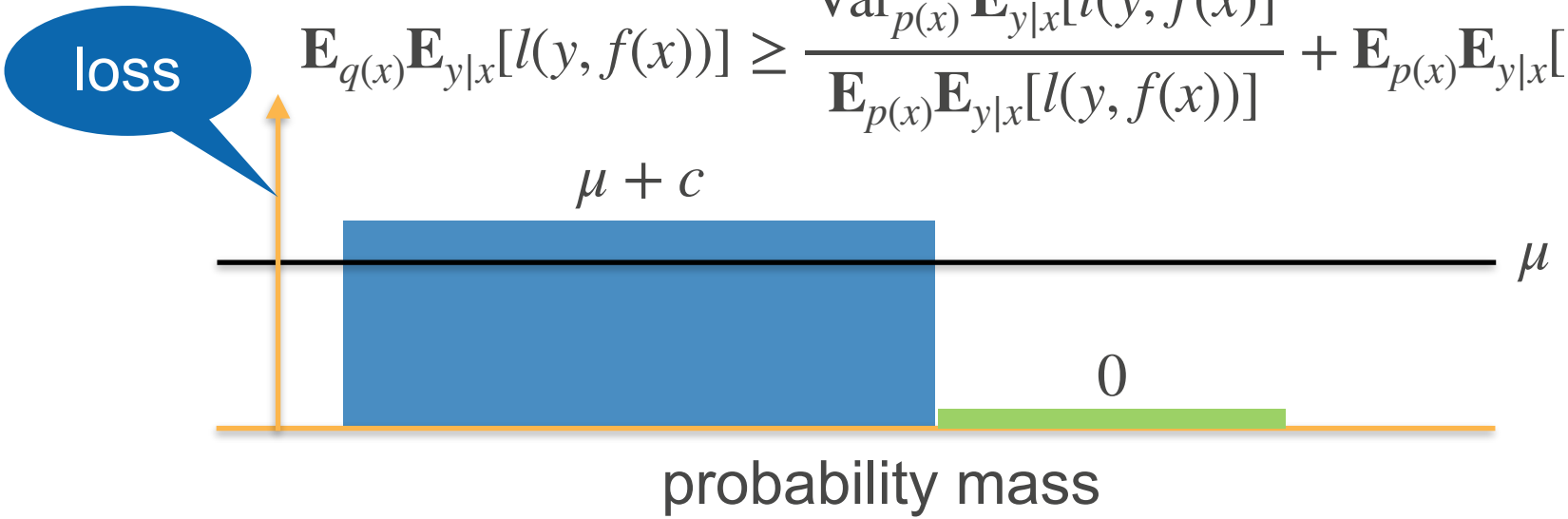
Train on IMDB

Test on weird prof

No Protection against Bias Theorem

- Estimator performs better (or worse) on some data
- We can always find a distribution q that is much worse

$$\mathbf{E}_{q(x)}\mathbf{E}_{y|x}[l(y, f(x))] \geq \frac{\mathrm{Var}_{p(x)}\,\mathbf{E}_{y|x}[l(y, f(x)]}{\mathbf{E}_{p(x)}\mathbf{E}_{y|x}[l(y, f(x))]} + \mathbf{E}_{p(x)}\mathbf{E}_{y|x}[l(y, f(x))]$$



TL;DR Testing where we have insufficient amounts of training data *may* yield strange results.

aws


LOGISTIC
REGRESSION

Recall - Multiclass Classification

Calibrated Scale

- Output matches probabilities (nonnegative, sums to 1)

$$p(y \mid o) = \text{softmax}(o)$$

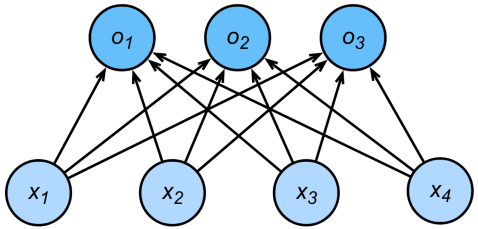
$$= \frac{\exp(o_y)}{\sum_i \exp(o_i)}$$

- Negative log-likelihood

$$-\log p(y \mid o) = \log \sum_i \exp(o_i) - o_y$$

Classification

- Multiple classes, typically multiple outputs
- Score *should* reflect confidence …



aws

Two Classes

- Classes 1 and -1

$$p(y = 1 \mid o) = \text{softmax}(o) = \frac{\exp(o_1)}{\exp(o_{-1}) + \exp(o_1)}$$

- Shift invariance $o_i \leftarrow o_i + c$

$$p(y = 1 \mid o) = \frac{\exp(o_1 + c)}{\exp(o_{-1} + c) + \exp(o_1 + c)} = \frac{\exp(o_1)}{\exp(o_{-1}) + \exp(o_1)}$$

Two Classes

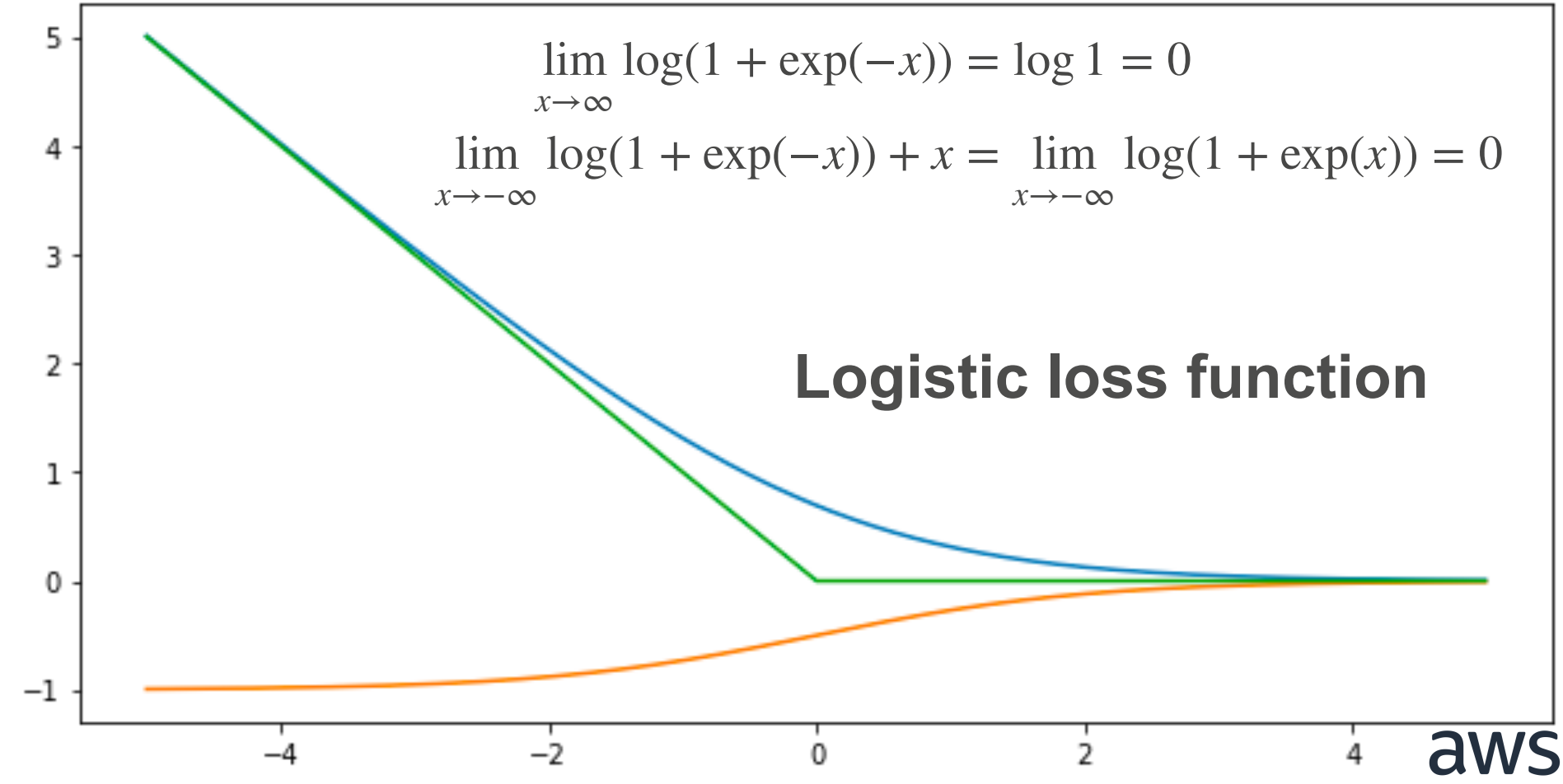
- Choose $o_{\_1} = 0$

$$p(y = 1 \mid o) = \frac{\exp(o_1)}{\exp(0) + \exp(o_1)} = \frac{1}{1 + \exp(-o_1)}$$

- Negative log-likelihood

$$-\log p(y \mid o) = \log(1 + \exp(-yo_1))$$

aws


$\lim_{x \to \infty} \log(1 + \exp(-x)) = \log 1 = 0$
$\lim_{x \to -\infty} \log(1 + \exp(-x)) + x = \lim_{x \to -\infty} \log(1 + \exp(x)) = 0$
Logistic loss function
5
4
3
2
1
0
−1
−4
−2
0
2
4

Logistic Regression Summary

- **Data**

$$(x_i, y_i) \text{ where } x_i \in \mathcal{X} \text{ and } y_i \in \{\pm 1\}$$

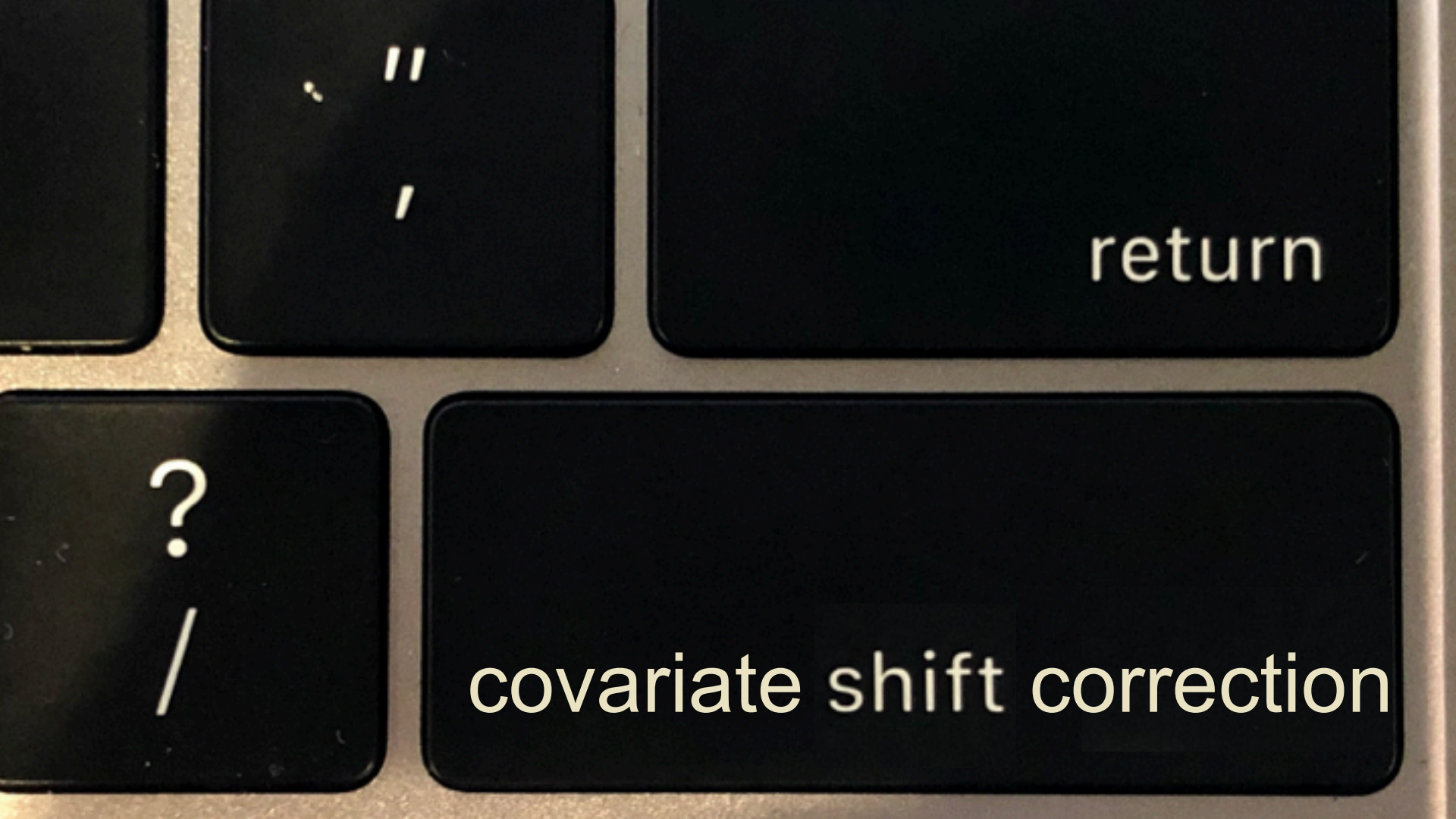
- **Objective**

$$\underset{w}{\text{minimize}} - \sum_{i=1}^{m} \log(1 + \exp(-y_i f(x_i, w))) + \text{penalty}(w)$$

- **Conditional Probability Estimate**

$$\log p(y = 1 \mid o) = \frac{1}{1 + \exp(-o)}$$

aws


"
'
return
?
/
covariate shift correction

Covariate shift correction

- Propensity scoring

$$\int dx\, q(x) f(x) = \int dx\, p(x) \underbrace{\frac{q(x)}{p(x)}}_{\alpha(x)} f(x) = \int dx\, p(x) \alpha(x) f(x)$$

- Need to find density ratio, but we don't have either one.
- Key idea: train a classifier between p and q

$$r(x, y) = \frac{1}{2}\left[p(x)\delta(y, 1) + q(x)\delta(y, -1)\right]$$

aws

Covariate shift correction

- Conditional class probability

$$r(y=1|x) = \frac{p(x)}{p(x)+q(x)} \text{ and hence } \alpha = \frac{q(x)}{p(x)} = \frac{r(y=-1|x)}{r(y=1|x)}$$

- Logistic regression

$$r(y=1\,|\,x) = \frac{1}{1+\exp(-f(x))}$$

$$\Longrightarrow \alpha(x) = \frac{r(y=-1\,|\,x)}{r(y=1\,|\,x)} = \exp(f(x))$$

aws

Covariate Shift Correction Redux

- Training and test data
- Split **as if it were a binary classification problem** (labels -1 and 1 for training and test respectively)
- Train with **logistic regression** to get f
- Use binary classifier output to reweight data
- Solve original problem but weighted

$$\sum_i l(x_i, y_i, g(x_i, w)) \longrightarrow \sum_i \exp(f(x_i)) \cdot l(x_i, y_i, g(x_i, w))$$

aws

Label shift

Training set








Training set

Test set

Why would anyone do this?

Label Shift

- **Medical diagnosis**
  - Train on data with few sick patients
  - Test on data during flu season where q(flu) > p(flu) while flu symptoms p(symptoms|flu) are still the same
- **Speech recognition**
  - Train on newscast data before election
  - Test on newscast after election (new topics, names, discussions, but still same language)

aws

Label Shift

$$q(x, y) = q(y)p(x|y)$$

- Data generating process p(x|y) is unchanged
- Labels change since the underlying cause changed

- Need to reweight according to $\beta(y) = \frac{q(y)}{p(y)}$ to get

$$\int q(y)\,dy \int p(x|y)\,dx\, l(f(x), y) = \int p(y)\frac{q(y)}{p(y)}\,dy \int p(x|y)\,dx\, l(f(x), y)$$

We don’t have samples from q(y)!

aws

Label Shift $q(x, y) = q(y)p(x|y)$

- **Key Idea - measure the estimates on test set**
  - p(x|y) is the same for training and test
  - Distribution of predictions |x,y has to be the same
- **Simple 'spectral' algorithm** (Lipton, Wang, Smola, 2018)
  - Confusion matrix $C[y'|y] = \Pr(\hat{y}(x) = y'|y)$ on hold out
  - Predicted label vector on test set $\mu[y'] = \Pr(\hat{y}(x) = y')$
  - Obtain q(y) via matrix inversion since

$$\mu[y'] = \sum_y C[y'|y]q(y)$$

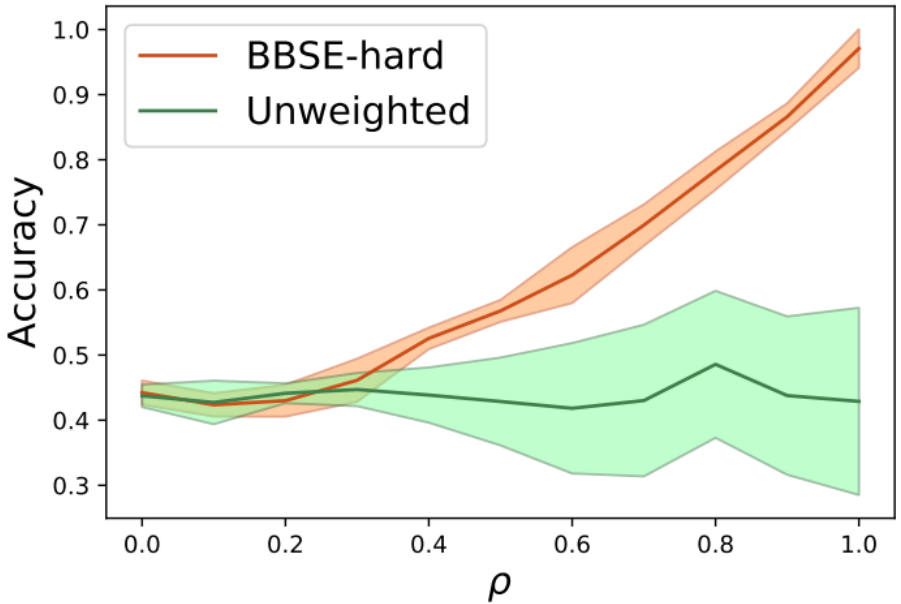
aws

Guarantees

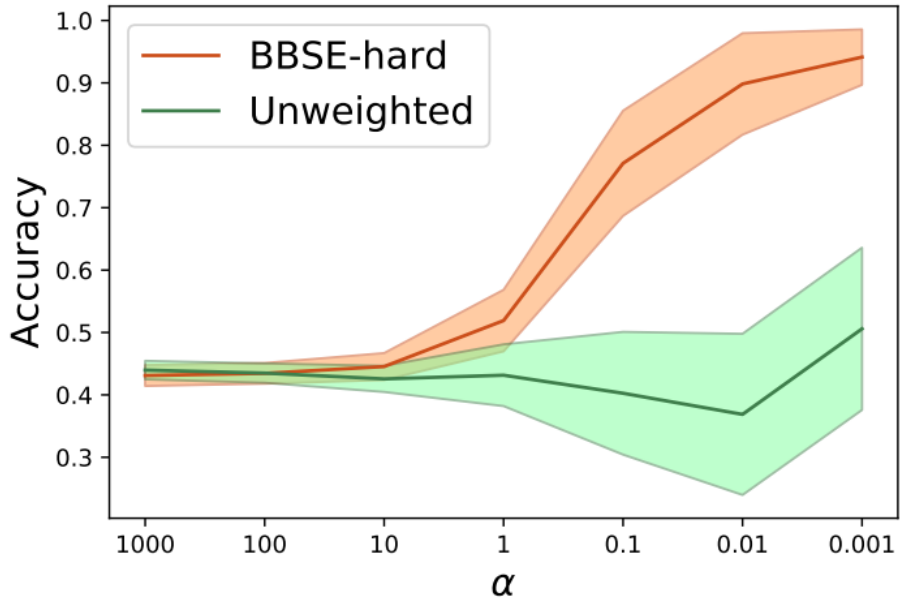
- **Robust under misspecification**
  - Even if the estimates y(x) are wrong, calibration is OK: (same errors on hold-out and test set)
  - Confusion matrix and label vector are concentrated: (use matrix Bernstein inequality)
- **Simple algorithm**
  - Cubic in number of classes, linear in sample size

aws

Black Box Shift Correction on CIFAR10



Tweaking one class probability



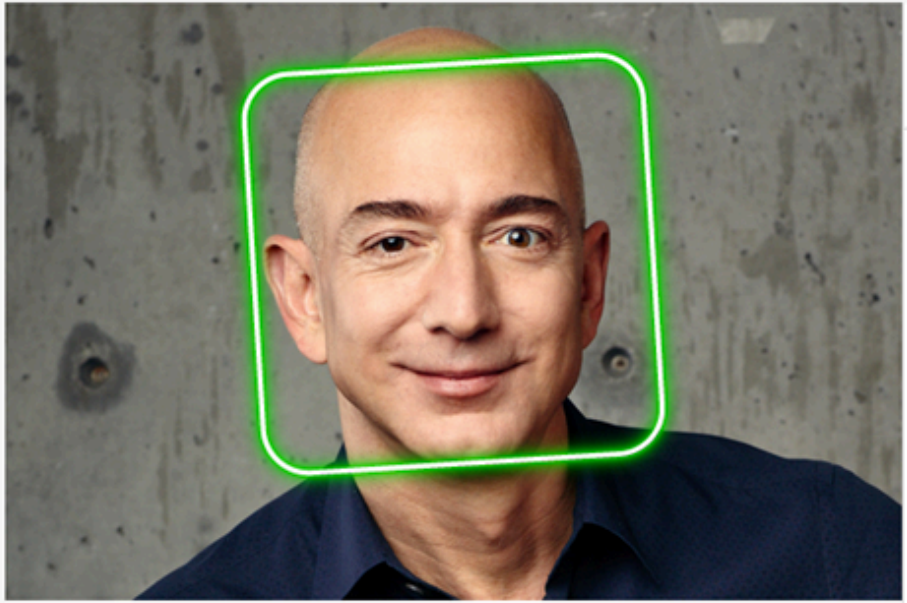
Dirichlet prior over shifts

aws

Adversarial data

Celebrity recognition

Rekognition automatically recognizes celebrities in images and provides confidence scores (Your images aren't stored.)



Done with the demo?
Download SDKs

▼ Results

Jeff Bezos
Learn More

Match confidence 100%

▶ Request

▶ Response

Choose a sample Image

Use your own image

Upload or drag and drop

Use image URL Go

aws

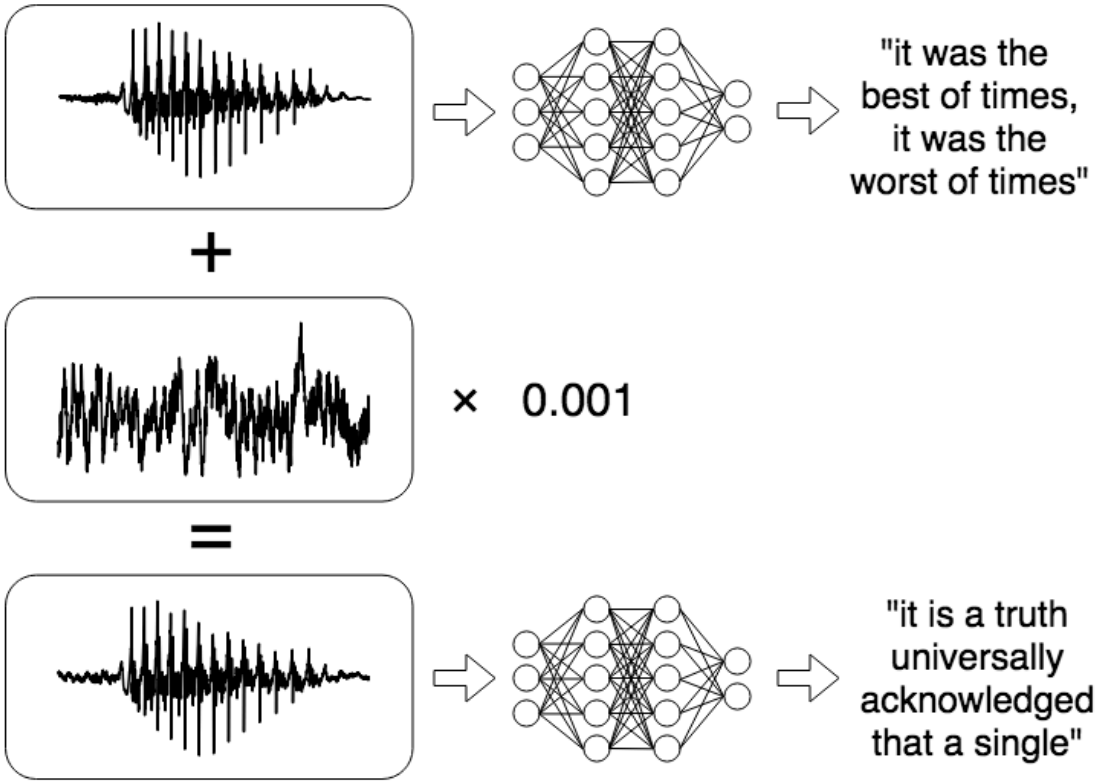
Adversarial Image Generation (e.g. Sharif et al. 2017)

Digital manipulation
to dodge recognition

In real life - via 3D
printed glasses

aws

Adversarial Audio Generation (e.g. Carlini & Wagner, 2018)



- Modify data slightly such as to obtain wrong class

$$\underset{\delta}{\text{maximize}}\ l(f(x+\delta), y)$$

$$\text{subject to } \|\delta\| \leq \epsilon$$

Different norms
Different datasets
Different papers …

aws

Why does this work?

‘Unnatural’ data

- Training and ‘natural’ test data live in small subset
- Adversarial data is slightly off that support
- Function behavior undefined away from where data occurs

aws

‘Unnatural’ data

- Training and ‘natural’ test data live in small subset
- Adversarial data is slightly off that support
- Function behavior undefined away from where data occurs

aws

Wow. Breathtaking. Is this new?

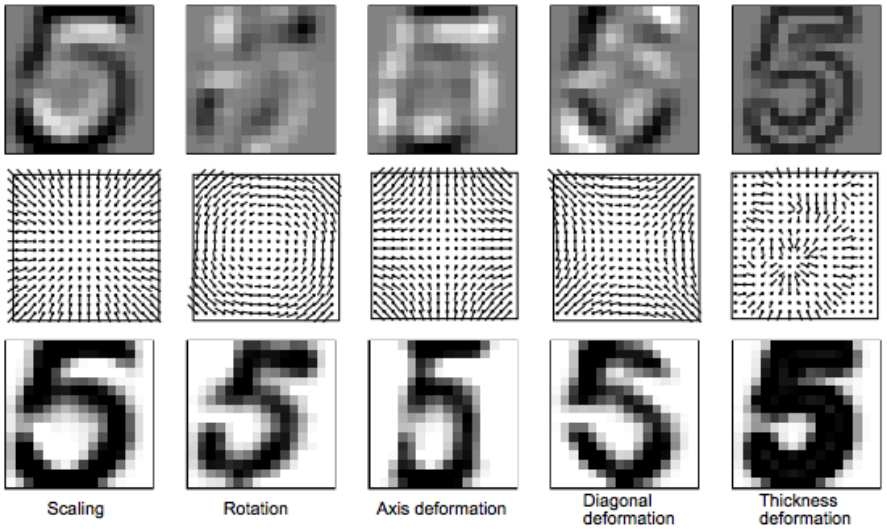
Spam defenses

- **While TRUE**
  - Mail host extends dataset and trains new classifier
  - Spammer’s e-mails are rejected
  - Spammer finds a modification that succeeds
- **Examples**
  - Add highly scoring words (or sentences) to email
  - Add highly scoring sentences (and vary them)
  - Change or forge header (‘Dear Alex, …’)

aws

Invariances

- Tangent Distance (Simard et al., 1995)
  - Invariance transforms don’t change the label
  - Explore data and their neighborhood



Invariances

- **Virtual Support Vectors** (Schoelkopf, 1997)
  Only change the data at the boundary (not enough RAM)
- **Data augmentation for training**
  - **Imagenet** (pretty much every paper)
    Cropping, scaling, change mean, per channel, …
  - **Speech Recognition**
    Background noise, scenes, …
  - **Document Analysis**
    Random substrings, word removal, insertion

aws

Invariant and robust loss

- **Convex loss** (Teo et al, 2005)
  - Family of transformations $\delta \in \Delta$
  - Penalty for extreme transformations $1 \geq \eta(\delta) \geq 0$
  - Find the 'worst' possible example at each step

Adversarially Robust Networks

$$L(x, y, f) = \sup_{\delta \in \Delta} \eta(\delta) l(f(x + \delta), y)$$

e.g. adversarial example generator Finds worst possible

Reduced penalty for extreme distortions

aws

Nonstationary Environments

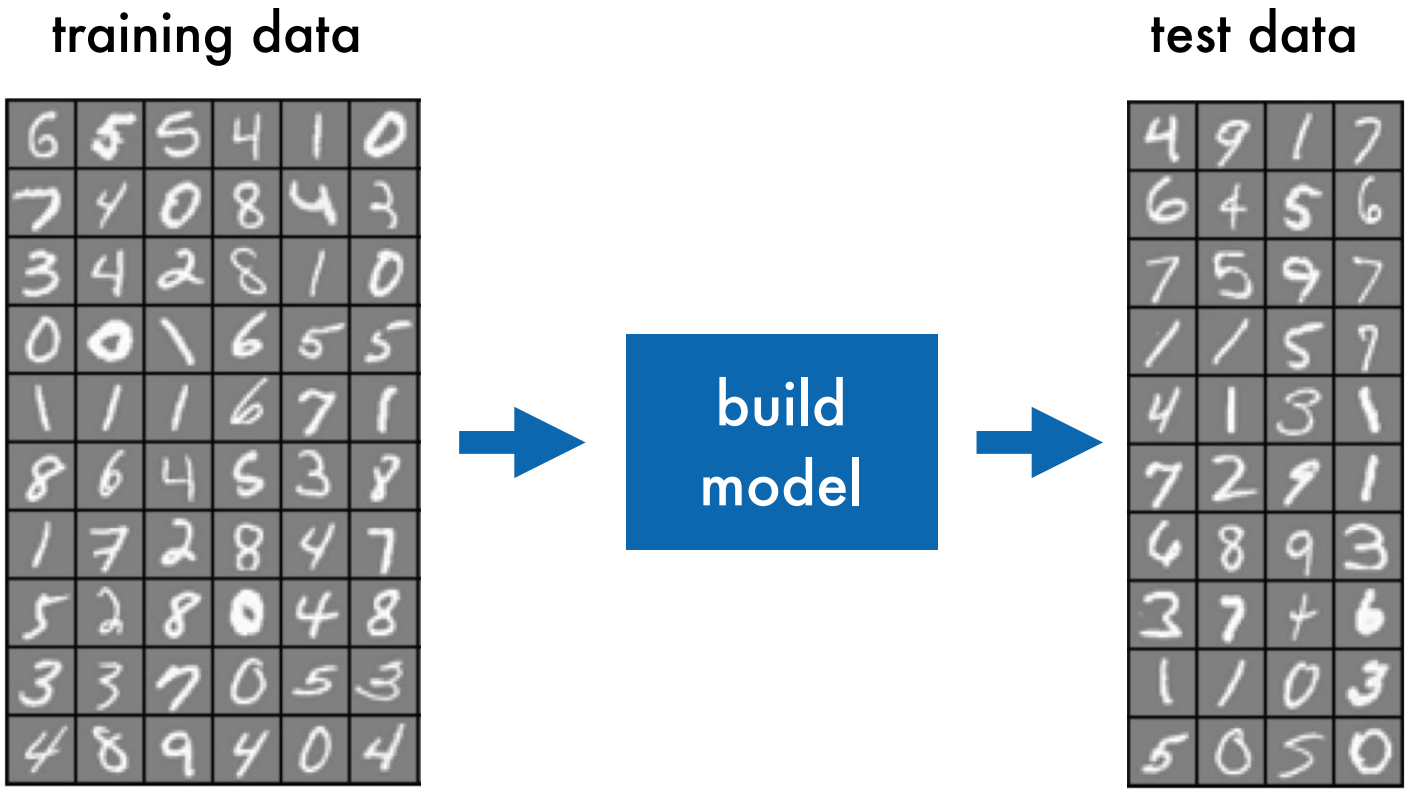
aws

Interaction with Environment

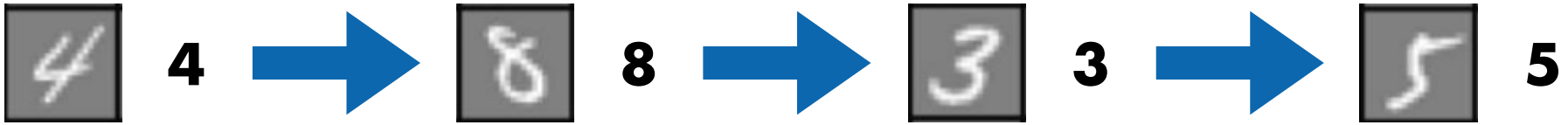
- **Batch** (download a book)
  Observe training data (x_1,y_1) ... (x_l,y_l) then deploy
- **Online** (follow the class)
  Observe x, predict f(x), observe y (stock market, homework)
- **Active learning** (ask questions in class)
  Query y for x, improve model, pick new x
- **Bandits** (do well at homework)
  Pick arm, get reward, pick new arm (also with context)
- **Reinforcement Learning** (play chess, drive a car)
  Take action, environment responds, take new action

aws

Batch



Online



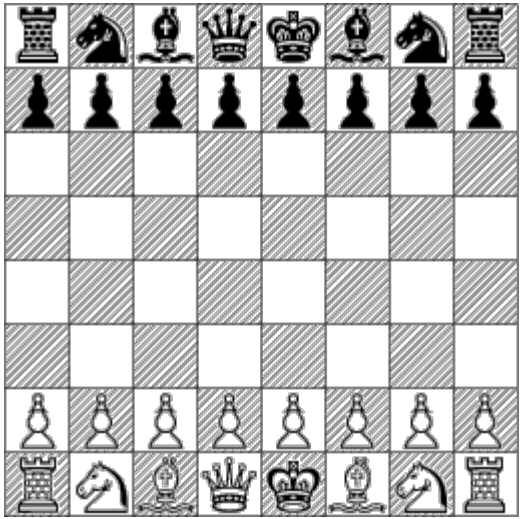
System improves as we see more data

Bandits

- Choose an arm (action)
- See what happens (get reward)
- Update model
- Choose next arm (action)
  The bandit **doesn't remember** what you did last summer.

Stateful Systems

Reinforcement Learning & Control



- Take action
- Environment reacts
- Observe stuff
- Update model

Repeat

- environment (cooperative, adversary, doesn't care)
- memory (goldfish, elephant)
- state space (tic tac toe, chess, car)
- past observations (server log, generated during training)

aws

Reinforcement Learning & Control

- **Games**
  - Chess, Go, Backgammon (fully observed)
  - Poker, Starcraft, ATARI (partially observed, random)
- **Parallelism**
  - Computation advertising, recommender systems (multiple agents & independent parallel games)
  - Load balancing & scheduling (multiple agents)
- **Actions**
  - Continuous decisions (driving, flying, robots in general, HVAC)
  - Discrete (elevator, work allocation)
- **Simulations**
  - MuJoCo style
  - Only reality (server center)

aws

Training ≠ Testing

- **Generalization performance**
  (the empirical distribution lies)
- **Covariate shift**
  (the covariate distribution lies)
- **Logistic regression**
  (tools to fix shift)
- **Covariate shift correction**
- **Label shift**
  (the label distribution lies)
- **Nonstationary Environments**

$$p_{\text{emp}}(x, y) \neq p(x, y)$$

$$p(x) \neq q(x)$$

$$\log(1 + \exp(-yf(x)))$$

$$\frac{1}{2}\left(p(x)\delta(1, y) + q(x)\delta(-1, y)\right)$$

$$p(y) \neq q(y)$$

aws

Numerical Stability

Wikipedia

Gradients for Neural Networks

- Consider a network with *d* layers

$$\mathbf{h}^t = f_t(\mathbf{h}^{t-1}) \quad \text{and} \quad y = \ell \circ f_d \circ \ldots \circ f_1(\mathbf{x})$$

- Compute the gradient of the loss ℓ w.r.t. $\mathbf{W}_t$

$$\frac{\partial \ell}{\partial \mathbf{W}^t} = \frac{\partial \ell}{\partial \mathbf{h}^d} \underbrace{\frac{\partial \mathbf{h}^d}{\partial \mathbf{h}^{d-1}} \cdots \frac{\partial \mathbf{h}^{t+1}}{\partial \mathbf{h}^t}} \frac{\partial \mathbf{h}^t}{\partial \mathbf{W}^t}$$

Multiplication of *d-t* matrices

aws

Two Issues for Deep Neural Networks

$$\prod_{i=t}^{d-1} \frac{\partial \mathbf{h}^{i+1}}{\partial \mathbf{h}^{i}}$$

Gradient Exploding

$$1.5^{100} \approx 4 \times 10^{17}$$

Gradient Vanishing

$$0.8^{100} \approx 2 \times 10^{-10}$$

aws

Example: MLP

- Assume MLP (without bias for simplicity)

$$f_t(\mathbf{h}^{t-1}) = \sigma(\mathbf{W}^t\mathbf{h}^{t-1}) \qquad \sigma \text{ is the activation function}$$

$$\frac{\partial \mathbf{h}^t}{\partial \mathbf{h}^{t-1}} = \text{diag}\left(\sigma'(\mathbf{W}^t\mathbf{h}^{t-1})\right)(W^t)^T \qquad \sigma' \text{ is the gradient function of } \sigma$$

$$\prod_{i=t}^{d-1} \frac{\partial \mathbf{h}^{i+1}}{\partial \mathbf{h}^i} = \prod_{i=t}^{d-1} \text{diag}\left(\sigma'(\mathbf{W}^i\mathbf{h}^{i-1})\right)(W^i)^T$$

aws

Gradient Exploding

- Use ReLU as the activation function

$$\sigma(x) = \max(0,x) \quad \text{and} \quad \sigma'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

- Elements of $\prod_{i=t}^{d-1} \frac{\partial \mathbf{h}^{i+1}}{\partial \mathbf{h}^i} = \prod_{i=t}^{d-1} \text{diag}\left(\sigma'(\mathbf{W}^i \mathbf{h}^{i-1})\right)(W^i)^T$ may from $\prod_{i=t}^{d-1} (W^i)^T$
  - Leads to large values when *d-t* is large

$$1.5^{100} \approx 4 \times 10^{17}$$

aws

Issues with Gradient Exploding

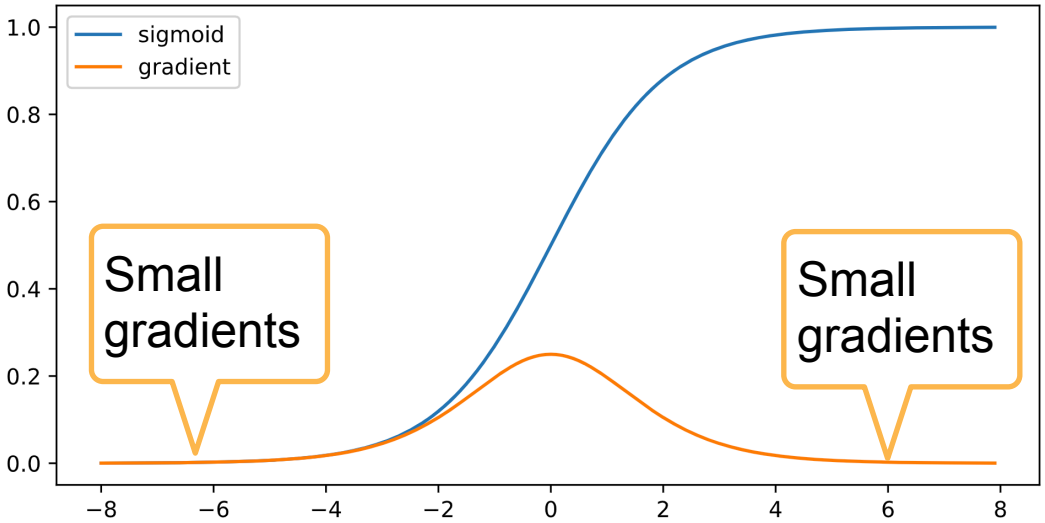
- Value out of range: infinity value
  - Severe for using 16-bit floating points
    - Range: 6e-5 - 6e4
- Sensitive to learning rate (LR)
  - Not small enough LR -> large weights -> larger gradients
  - Too small LR -> No progress
  - May need to change LR dramatically during training

aws

Gradient Vanishing

- Use sigmoid as the activation function

$$\sigma(x) = \frac{1}{1 + e^{-x}} \qquad \sigma'(x) = \sigma(x)(1 - \sigma(x))$$



Gradient Exploding

- Use sigmoid as the activation function

$$\sigma(x) = \frac{1}{1 + e^{-x}} \qquad \sigma'(x) = \sigma(x)(1 - \sigma(x))$$

- Elements $\prod_{i=t}^{d-1} \frac{\partial \mathbf{h}^{i+1}}{\partial \mathbf{h}^{i}} = \prod_{i=t}^{d-1} \text{diag}\left(\sigma'(\mathbf{W}^i \mathbf{h}^{i-1})\right)(W^i)^T$ are products of *d-t* small values

$$0.8^{100} \approx 2 \times 10^{-10}$$

aws

Issues with Gradient Vanishing

- Gradients with value 0
  - Severe with 16-bit floating points
- No progress in training
  - No matter how to choose learning rate
- Severe with bottom layers
  - Only top layers are well trained
  - No benefit to make networks deeper

aws

Stabilize Training

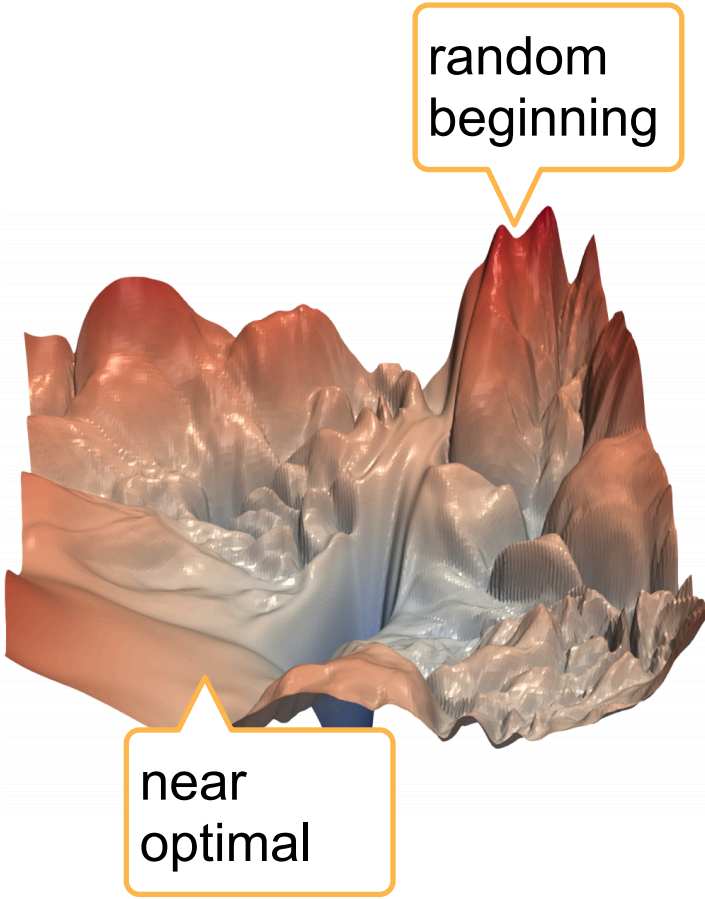
Stabilize Training

- Goal: make sure gradient values are in a proper range
  - E.g. in [1e-6, 1e3]
- Multiplication -> plus
  - ResNet, LSTM
- Normalize
  - Batch Normalization, Gradient clipping
- Proper weight initialization and activation functions

aws

Weight Initialization

- Initialize weights with random values in a proper range
- The beginning of training easily suffers to numerical instability
  - The surface far away from an optimal can be complex
  - Near optimal may be flatter
- Initializing according to $\mathcal{N}(0, 0.01)$ works well for small networks, but not guarantee for deep neural networks



Constant Variance for each Layer

- Treat both layer outputs and gradients are random variables
- Make the mean and variance for each layer's output are same, similar for gradients

Forward

$$\mathbb{E}[h_i^t] = 0$$

$$\mathrm{Var}[h_i^t] = a$$

Backward

$$\mathbb{E}\left[\frac{\partial \ell}{\partial h_i^t}\right] = 0 \quad \mathrm{Var}\left[\frac{\partial \ell}{\partial h_i^t}\right] = b \qquad \forall i, t$$

a and *b* are constants

aws

Example: MLP

- Assumptions
  - i.i.d $w_{i,j}^t$, $\mathbb{E}[w_{i,j}^t] = 0, \mathrm{Var}[w_{i,j}^t] = \gamma_t$
  - h_i^{t-1} is independent to $w_{i,j}^t$
  - identity activation: $\mathbf{h}^t = \mathbf{W}^t \mathbf{h}^{t-1}$ with $\mathbf{W}^t \in \mathbb{R}^{n_t \times n_{t-1}}$

$$\mathbb{E}[h_i^t] = \mathbb{E}\left[\sum_j w_{i,j}^t h_j^{t-1}\right] = \sum_j \mathbb{E}[w_{i,j}^t]\mathbb{E}[h_j^{t-1}] = 0$$

aws

Forward Variance

$$\mathrm{Var}[h_i^t] = \mathbb{E}[(h_i^t)^2] - \mathbb{E}[h_i^t]^2 = \mathbb{E}\left[\left(\sum_j w_{i,j}^t h_j^{t-1}\right)^2\right]$$

$$= \mathbb{E}\left[\sum_j \left(w_{i,j}^t\right)^2 \left(h_j^{t-1}\right)^2 + \sum_{j \neq k} w_{i,j}^t w_{i,k}^t h_j^{t-1} h_k^{t-1}\right] \quad \Rightarrow \quad n_{t-1}\gamma_t = 1$$

$$= \sum_j \mathbb{E}\left[\left(w_{i,j}^t\right)^2\right] \mathbb{E}\left[\left(h_j^{t-1}\right)^2\right]$$

$$= \sum_j \mathrm{Var}[w_{i,j}^t]\mathrm{Var}[h_j^{t-1}] = n_{t-1}\gamma_t \mathrm{Var}[h_j^{t-1}]$$

aws

Backward Mean and Variance

- Apply forward analysis as well

$$\frac{\partial \ell}{\partial \mathbf{h}^{t-1}} = \frac{\partial \ell}{\partial \mathbf{h}^{t}} \mathbf{W}^{t} \quad \text{leads to} \quad \left(\frac{\partial \ell}{\partial \mathbf{h}^{t-1}} \right)^{T} = (W^{t})^{T} \left(\frac{\partial \ell}{\partial \mathbf{h}^{t}} \right)^{T}$$

$$\mathbb{E}\left[\frac{\partial \ell}{\partial h_i^{t-1}} \right] = 0$$

$$\mathrm{Var}\left[\frac{\partial \ell}{\partial h_i^{t-1}} \right] = n_t \gamma_t \mathrm{Var}\left[\frac{\partial \ell}{\partial h_j^{t}} \right] \quad \Rightarrow \quad n_t \gamma_t = 1$$

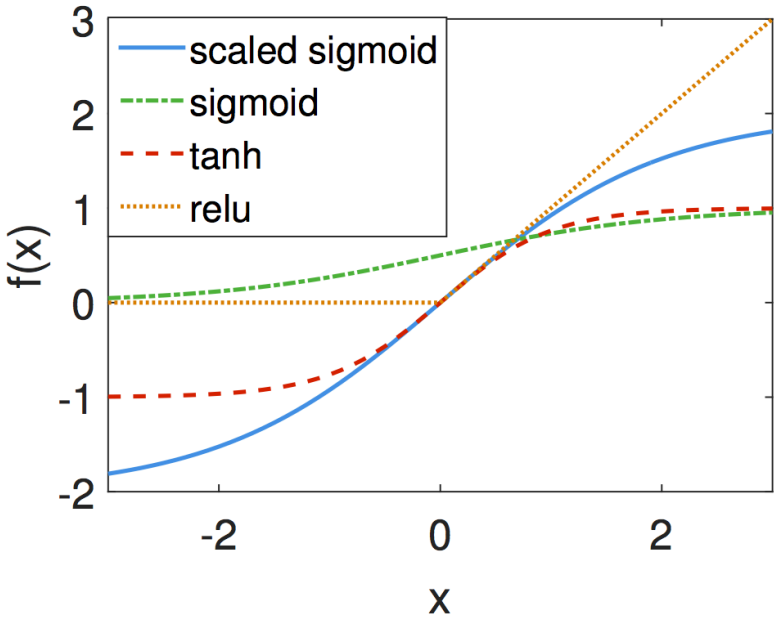
aws

Xavier Initialization

- Conflict goal to satisfies both $n_{t-1}\gamma_t = 1$ and $n_t\gamma_t = 1$
- Xavier $\gamma_t(n_{t-1} + n_t)/2 = 1 \quad \rightarrow \quad \gamma_t = 2/(n_{t-1} + n_t)$
  - Normal distribution $\mathcal{N}\left(0, \sqrt{2/(n_{t-1} + n_t)}\right)$
  - Uniform distribution $\mathcal{U}\left(-\sqrt{6/(n_{t-1} + n_t)}, \sqrt{6/(n_{t-1} + n_t)}\right)$
    - Variance of $\mathcal{U}[-a, a]$ is $a^2/3$
- Adaptive to weight shape, especially when n_t varies

aws

Activation



A Simple Linear Activation Function

- Assume $\sigma(x) = \alpha x + \beta$

$$\mathbf{h}' = \mathbf{W}^t \mathbf{h}^{t-1} \quad \text{and} \quad \mathbf{h}^t = \sigma(\mathbf{h}')$$

$$\mathbb{E}[h_i^t] = \mathbb{E}\left[\alpha h_i' + \beta\right] = \beta \quad \Rightarrow \quad \beta = 0$$

$$\begin{aligned}
\mathrm{Var}[h_i^t] &= \mathbb{E}[(h_i^t)^2] - \mathbb{E}[h_i^t]^2 \\
&= \mathbb{E}[(\alpha h_i' + \beta)^2] - \beta^2 \quad \Rightarrow \quad \alpha = 1 \\
&= \mathbb{E}[\alpha^2 (h_i')^2 + 2\alpha\beta h_i' + \beta^2] - \beta^2 \\
&= \alpha^2 \mathrm{Var}[h_i']
\end{aligned}$$

aws

Backward

- Assume $\sigma(x) = \alpha x + \beta$

$$\frac{\partial \ell}{\partial \mathbf{h}'} = \frac{\partial \ell}{\partial \mathbf{h}^t}(W^t)^T \quad \text{and} \quad \frac{\partial \ell}{\partial \mathbf{h}^{t-1}} = \alpha \frac{\partial \ell}{\partial \mathbf{h}'}$$

$$\mathbb{E}\left[\frac{\partial \ell}{\partial h_i^{t-1}}\right] = 0 \quad \Rightarrow \quad \beta = 0$$

$$\mathrm{Var}\left[\frac{\partial \ell}{\partial h_i^{t-1}}\right] = \alpha^2 \mathrm{Var}\left[\frac{\partial \ell}{\partial h_j'}\right] \quad \Rightarrow \quad \alpha = 1$$

aws

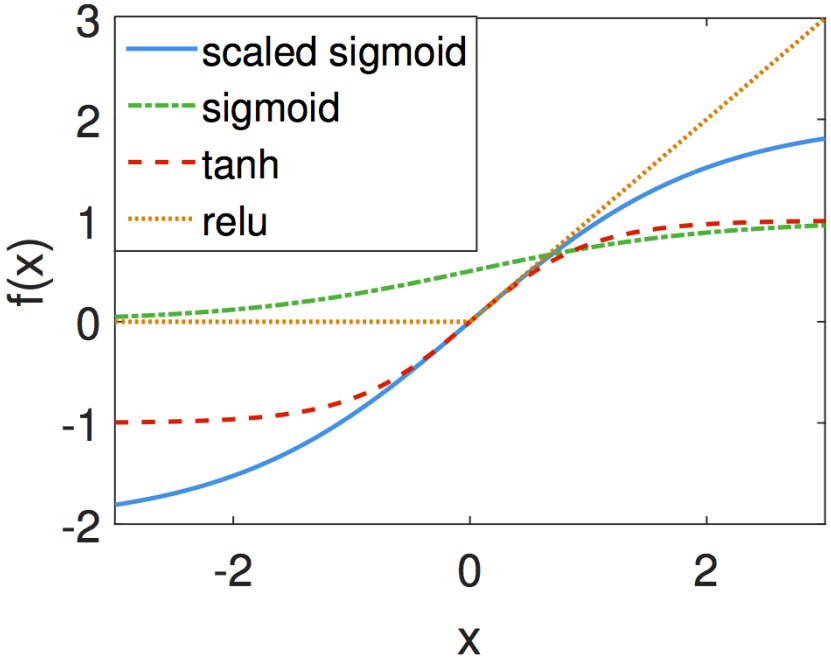
Revise Activation Functions

- By the Taylor expansions

$$\text{sigmoid}(x) = \frac{1}{2} + \frac{x}{4} - \frac{x^3}{48} + O(x^5)$$

$$\tanh(x) = 0 + x - \frac{x^3}{3} + O(x^5)$$

$$\text{relu}(x) = 0 + x \quad \text{for } x \geq 0$$

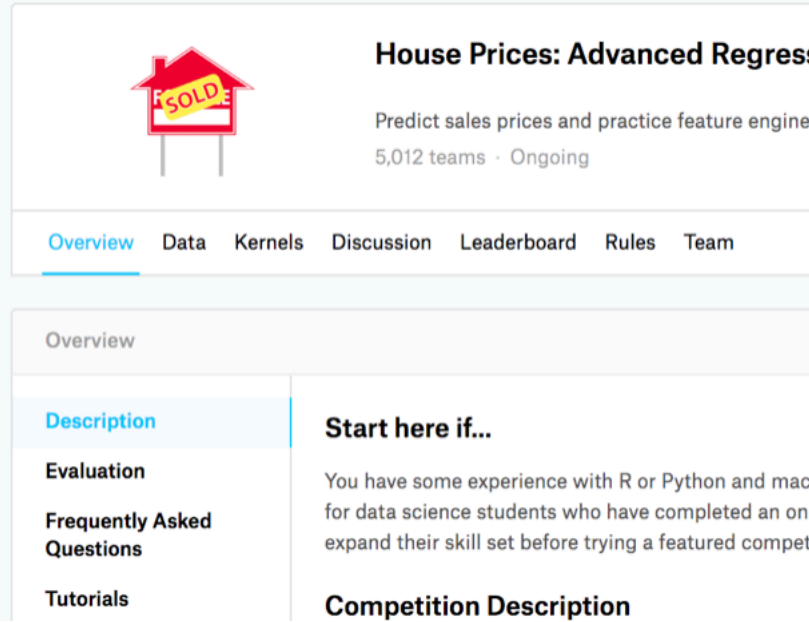


- "Correct" sigmoid by

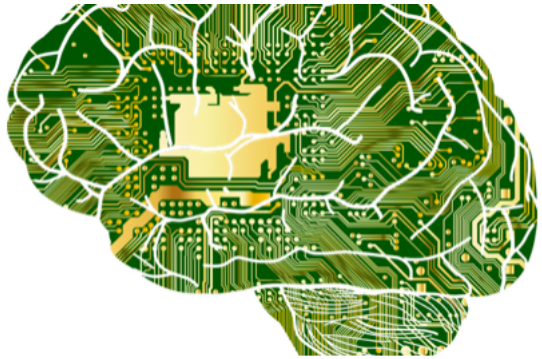
$$4 \times \text{sigmoid}(x) - 2$$

aws

Predicting House Prices on Kaggle



Hardware for Deep Learning

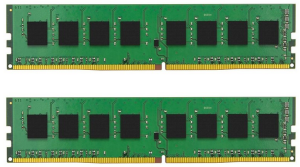


(articleshub360.com)

aws

Your GPU Computer

Intel i7
0.15 TFLOPS

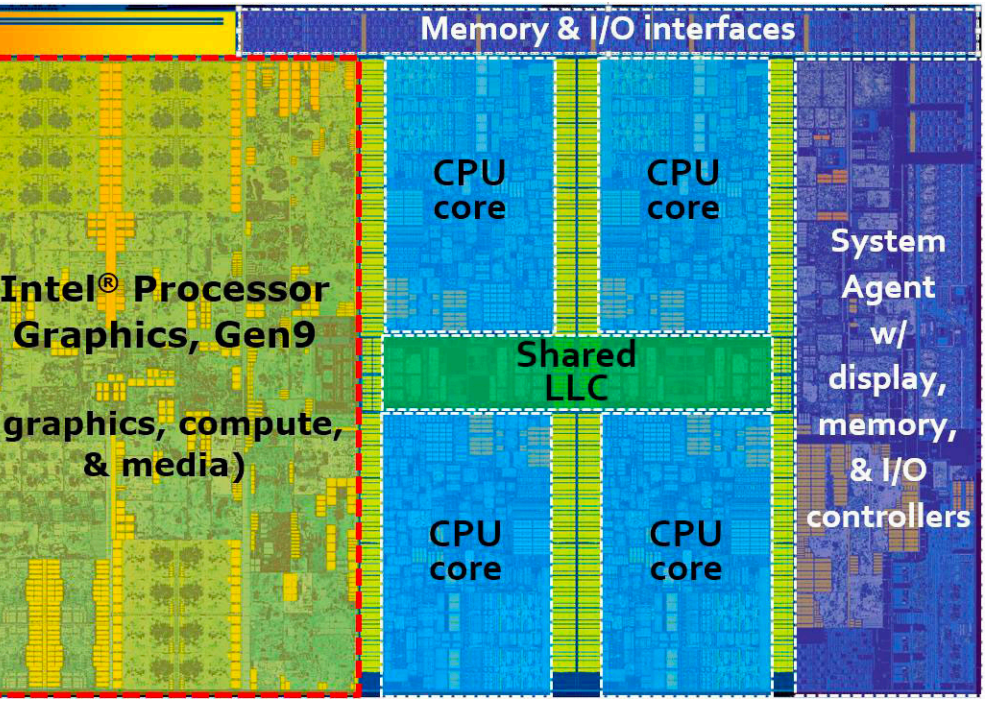


DDR4
32 GB

Nvidia Titan X
12 TFLOPS
16 GB

aws

Intel i7-6700K



- 4 Physical cores
- Per core
  - 64KB L1 cache
  - 256KB L2 cache
- Shared 8MB L3 cache
- 30 GB/s to main memory

aws

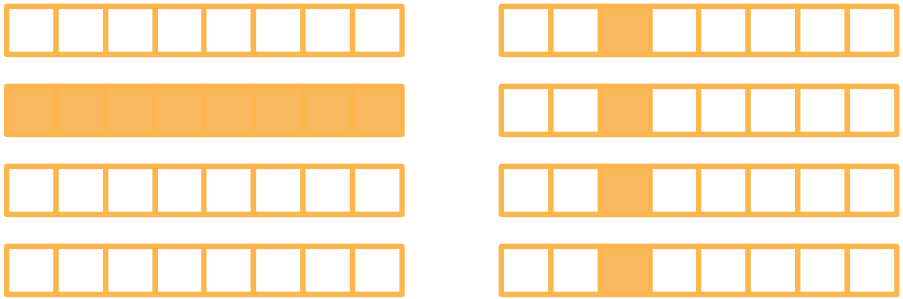
Improve CPU Utilization I

- Before computing *a* + *b*, need to prepare data first
  - Main memory -> L3 -> L2 -> L1 -> registers
    - L1 cache reference time: 0.5 ns
    - L2 cache reference time 7 ns (14 x L1)
    - Main memory reference time 100ns (200 x L1)
- Improve temporal and spatial memory locality
  - Temporal: reuse data so we keep them on cache
  - Spatial: read data sequential so we can pre-fetch data

aws

Case Study

- For a matrix stored in raw-major, accessing a column is slower than accessing a row
  - CPU reads 64 bytes (cache line) each time
  - CPU “smartly” reads the next cache line ahead when it’s needed



aws

Improve CPU Utilization II

- Server CPUs may have dozens of cores
  - EC2 P3.16xlarge: 2 Intel Xeon CPUs, 32 physical cores in total
- Parallelization to use all cores
  - Hyper-threading may not help because of shared registers

aws

Case Study

- Left is slower than right (Python)

```
for i in range(len(a)):
    c[i] = a[i] + b[i]
```

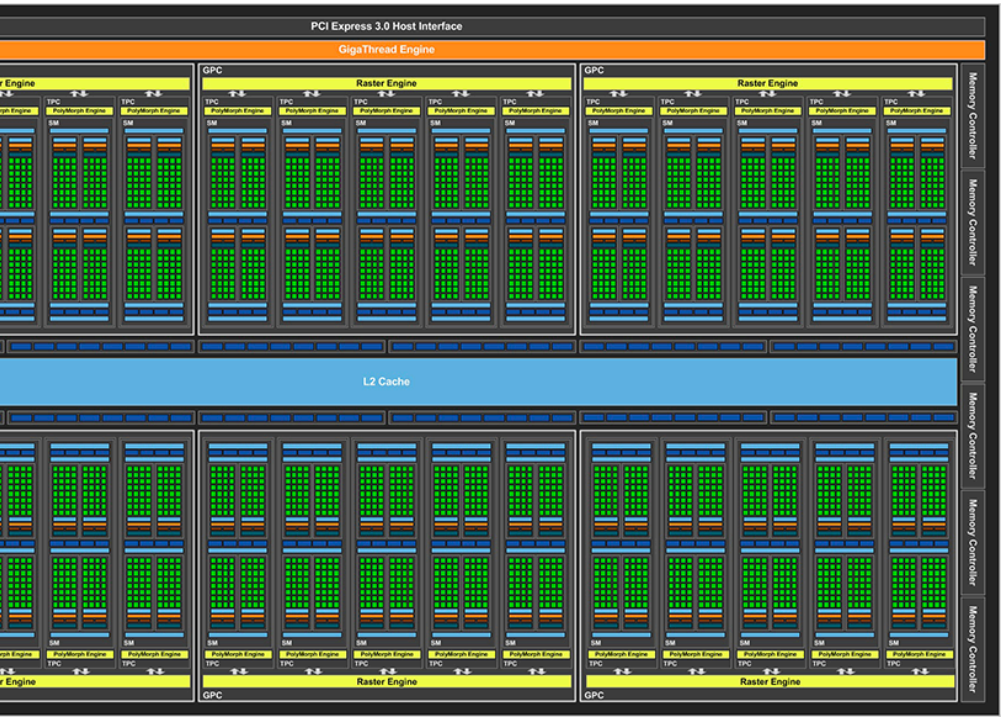
```
c = a + b
```

  - Left issues *n* calls, each invoking has certain overhead (e.g. several microsecond)
  - Right is easier to be parallelized by a C++ operator

```
#pragma omp for
for (i=0; i<a.size(); i++) {
    c[i] = a[i] + b[i]
}
```


aws

Nvidia Titan X (Pascal)



- 2584 cores, each core
  - Multiple arithmetic units
- 3MB L2 cache
- 480 GB/s memory bandwidth

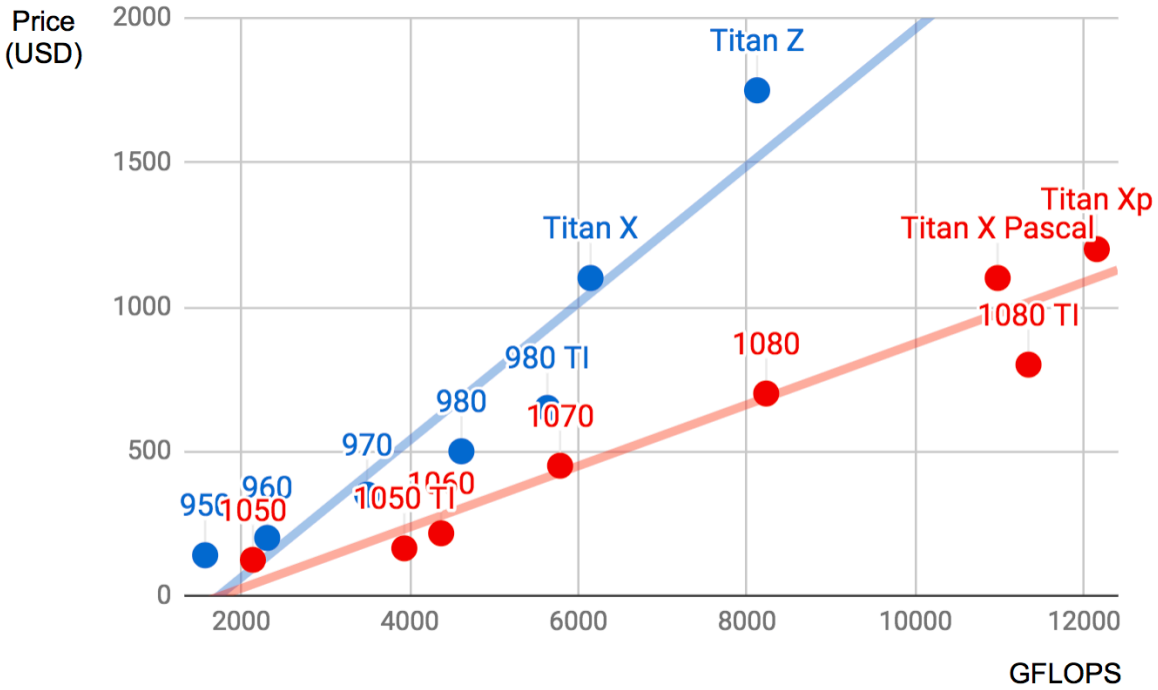
aws

Improve GPU Utilization

- Parallelization
  - Uses thousands of threads
- Memory locality
  - Simple cache architecture and small cache size
- Simple control flows
  - Very limited support
  - Large synchronization

aws

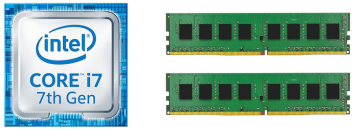
Buy GPUs



- Each series improve the previous one
  - Buy new models
- Price is linear to power in a series

aws

CPU vs GPU



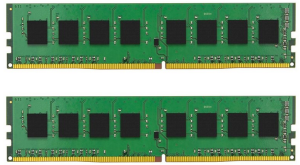
# Cores	6 / 64	2K / 4K
TFLOPS	0.2 / 1	10 / 100
Memory size	32 GB / 1 TB	16 GB / 32 GB
Memory bandwidth	30 GB/s / 100 GB/s	400 GB/s / 1 TB/s
Control flow	Strong	Weak

Typical / High End

aws

CPU/GPU Bandwidth

30 GB/s



PCIe 3.0 16x: 16 GB/s

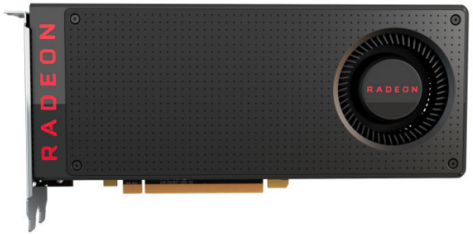
480 GB/s

- Do not copy data between GPU and CPU frequently
  - Limited PCIe bandwidth
  - Synchronization overhead

aws

More CPUs and GPUs

- CPU
  - Desktop/Server: AMD
  - Edge: ARM
- GPU
  - Desktop/Server GPU: AMD, Intel
  - Edge: ARM, Qualcomm



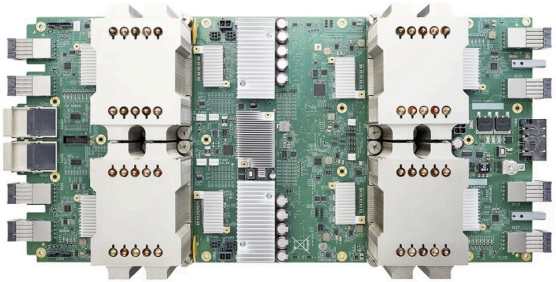
aws

Programming on CPU/GPU

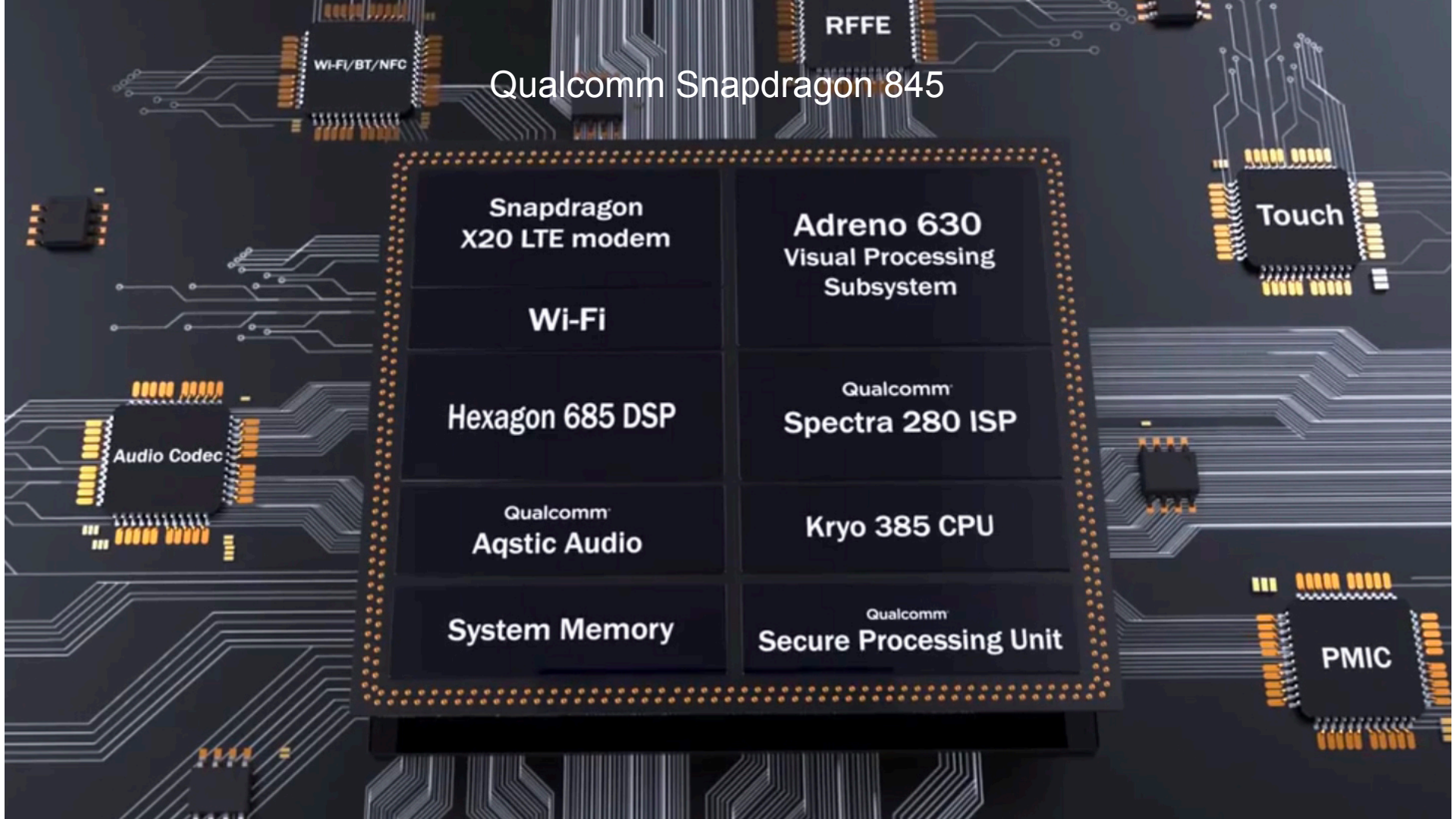
- CPU: C++ or any other high-performance language
  - Mature compilers with performance guaranteed
- GPU
  - CUDA for Nvidia
    - Rich features, mature complier and driver
  - OpenCL for others
    - Quality varies for chip vendors

aws

More Promising Hardware



aws


Qualcomm Snapdragon 845
Wi-Fi/BT/NFC
RFFE
Touch
Audio Codec
PMIC
Snapdragon
X20 LTE modem
Wi-Fi
Adreno 630
Visual Processing
Subsystem
Hexagon 685 DSP
Qualcomm
Spectra 280 ISP
Qualcomm
Aqstic Audio
Kryo 385 CPU
System Memory
Qualcomm
Secure Processing Unit

Digital Signal Processor

- Designed for digital processing algorithms
  - Dot product, Convolution, FFT
- Low power and high performance
  - 5x faster than mobile GPUs, uses less power
- VLIW: Very long instruction word
  - Hundreds multiply-accumulates in a single instruction
- Hard to program and debug
- Compiler toolchain quality varies from chip vendors

aws

Field-Programmable Gate Array (FPGA)

- Contains a large number of programmable logic blocks and reconfigurable interconnects
- Can be configured to perform complex functions
  - Common languages: VHDL and Verilog
- Often has high efficiency than general purpose hardware
- Toolchain qualities vary
- Each “compilation” may take several hours

aws

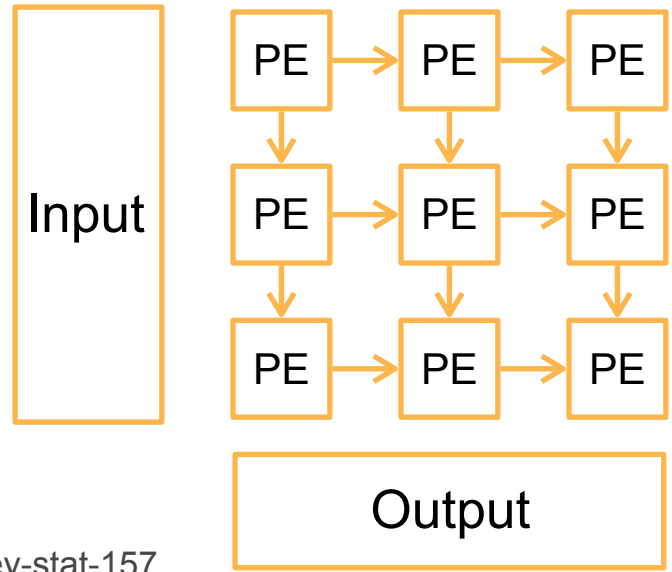
AI ASIC

- Hot topic in deep learning
  - Every major company is building their chips (Intel, Qualcomm, Google, Amazon, Facebook, …)
- Google TPU is a landmark chip
  - Matches high-end Nvidia GPU's performance, but 10x-100x cheaper
  - Widely deployed in Google
  - The core is a systolic array

aws

Systolic Array

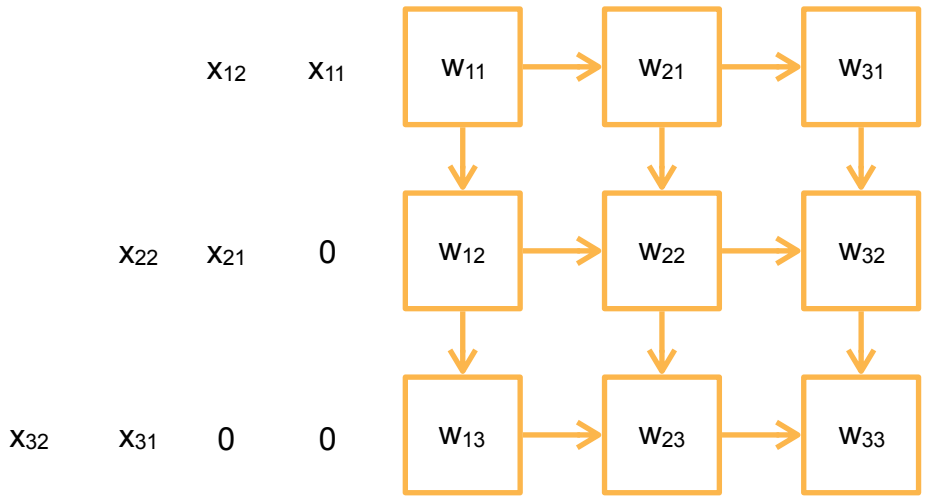
- An array of processing elements (PE)
- Good at performing matrix-matrix multiplication
- Relative easy/cheaper to build



aws

Matrix Multiplication with Systolic Array

Time 0

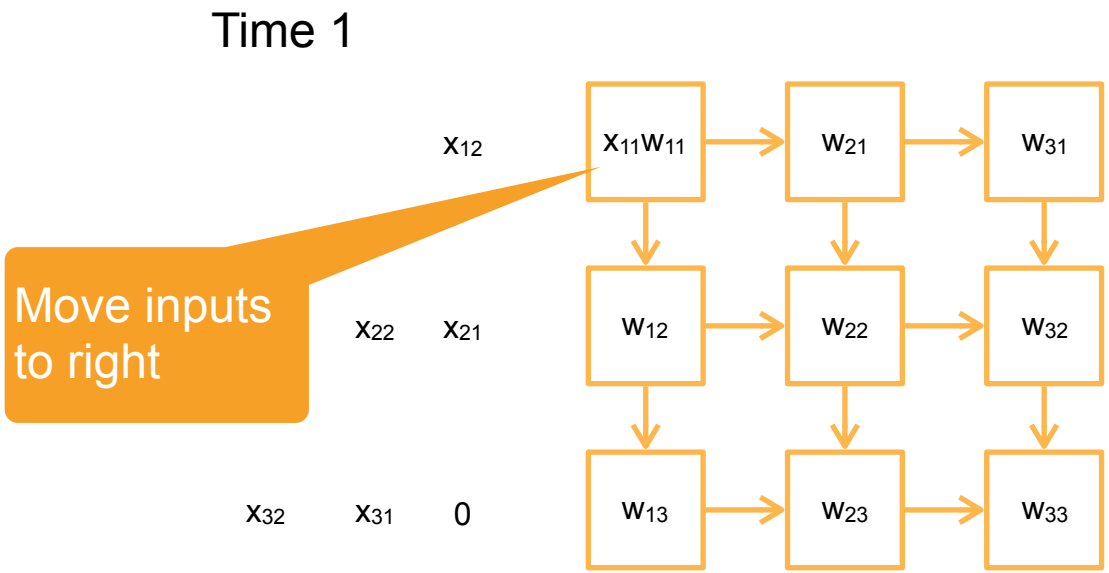


$$\mathbf{Y} = \mathbf{WX}$$

3x2 3x3 3x2

aws

Matrix Multiplication with Systolic Array

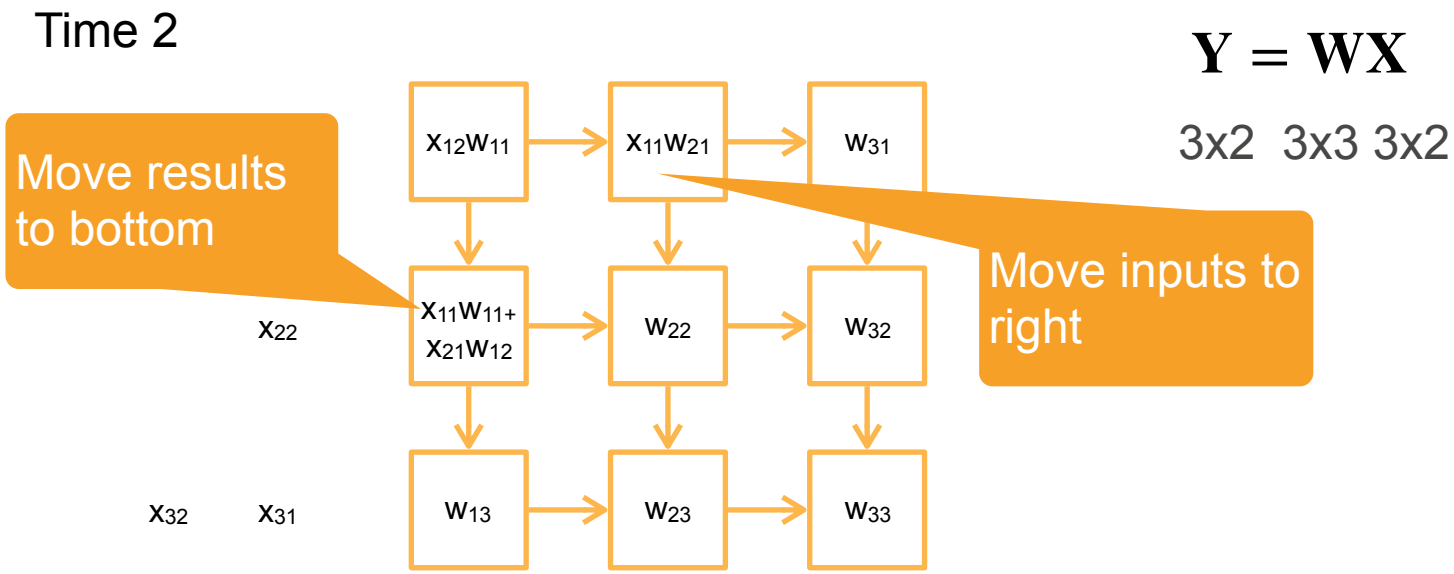


$$\mathbf{Y} = \mathbf{WX}$$

3x2 3x3 3x2

aws

Matrix Multiplication with Systolic Array

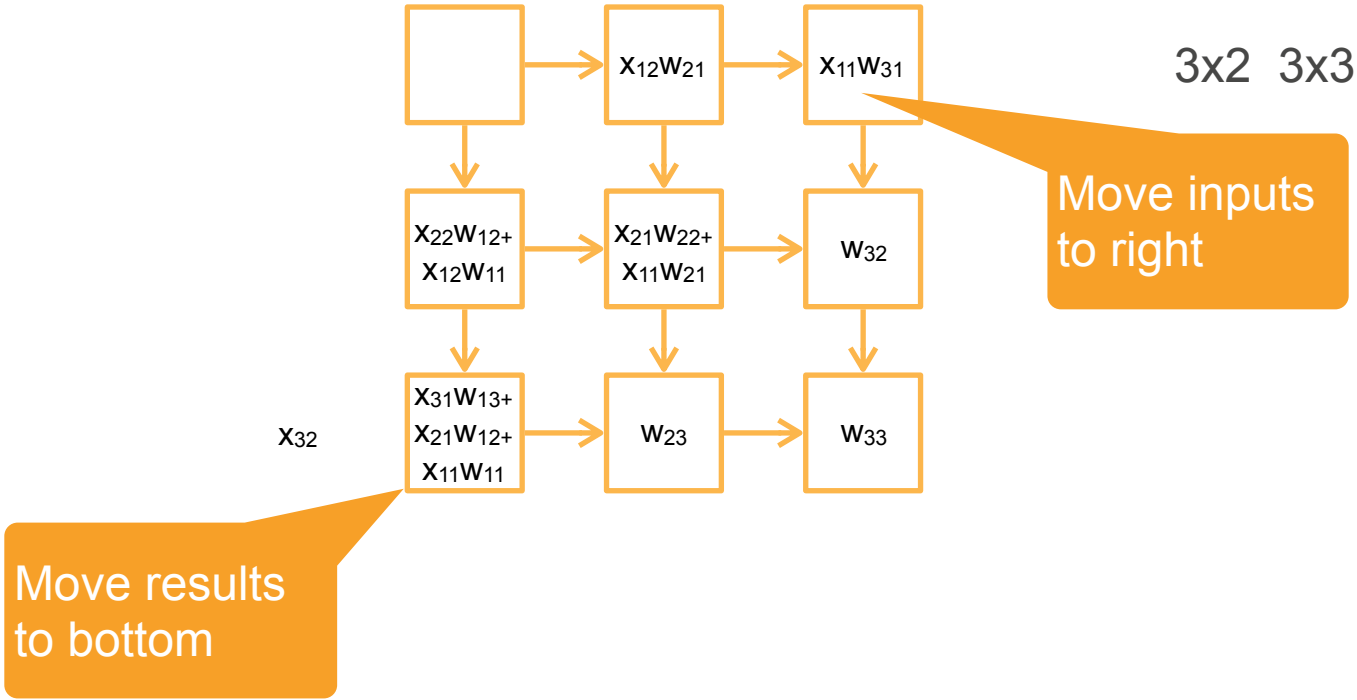


Matrix Multiplication with Systolic Array

Time 3

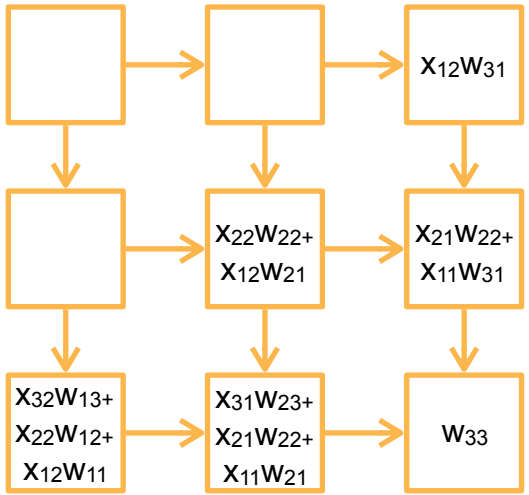
$$\mathbf{Y} = \mathbf{WX}$$

3x2 3x3 3x2



Matrix Multiplication with Systolic Array

Time 4



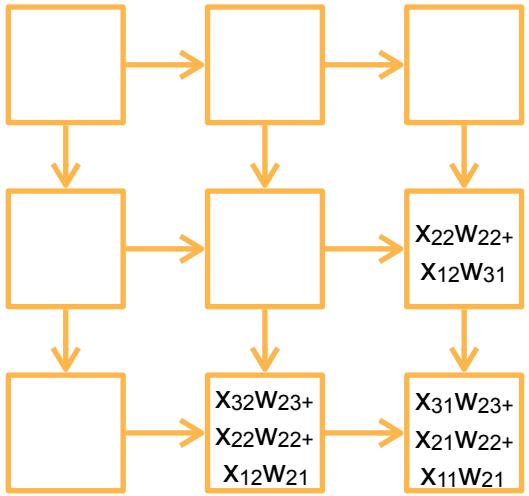
$$\mathbf{Y} = \mathbf{WX}$$

3x2 3x3 3x2

aws

Matrix Multiplication with Systolic Array

Time 5



y_{12} y_{21}

y_{11}

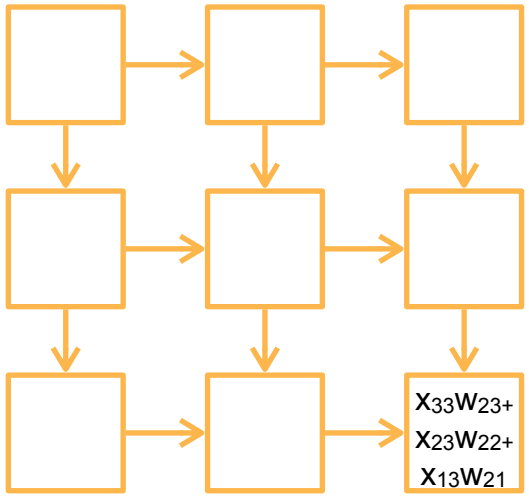
$$\mathbf{Y} = \mathbf{WX}$$

3x2 3x3 3x2

aws

Matrix Multiplication with Systolic Array

Time 6



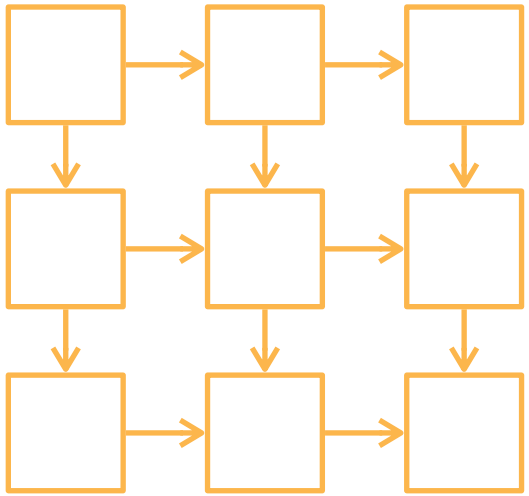
$$\mathbf{Y} = \mathbf{WX}$$

3x2 3x3 3x2

aws

Matrix Multiplication with Systolic Array

Time 7



$$\mathbf{Y} = \mathbf{WX}$$

3x2 3x3 3x2

y_{32}

y_{22} y_{31}

y_{12} y_{21}

y_{11}

aws

Systolic Array

- For general size matrix multiplication, slice and pad inputs to match the SA size
- Batch inputs to reduce the latency cost
- ASIC has other dedicated components for other NN operations, such as sigmoid

aws


Flexibility and Ease of Use
intel
CORE i7
7th Gen
TITAN X
Hexagon 685 DSP
Xilinx®
FPGA
Performance & Power Efficiency