End of Days

What was this course about?

- Machine Learning
- Deep Learning
- Really just thinking, the process

Along the way we

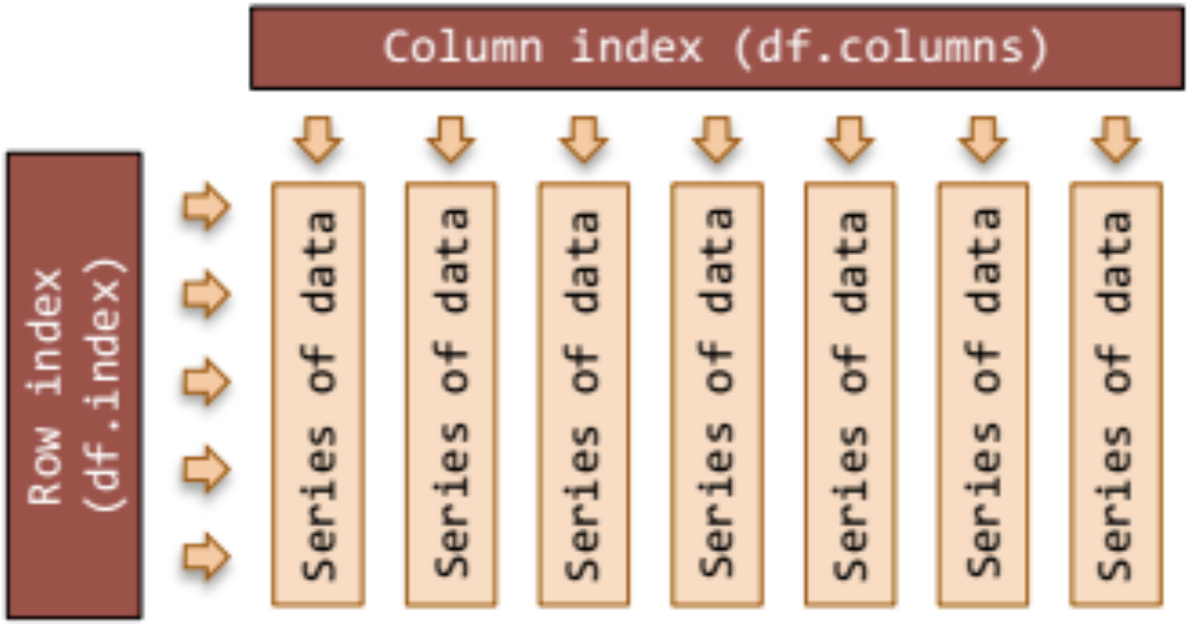
- learn how to create and regularize models
- learn how to optimize objective functions such as loss functions using Stochastic Gradient Descent
- learnt how to create (simple) archirectures to solve problems
- learnt how to transfer from one model to the next

Concepts running through:

- Fitting parameters vs hyperparameters
- Regularization
- Validation testing
- Stochastic Gradient Descent
- Learning representations

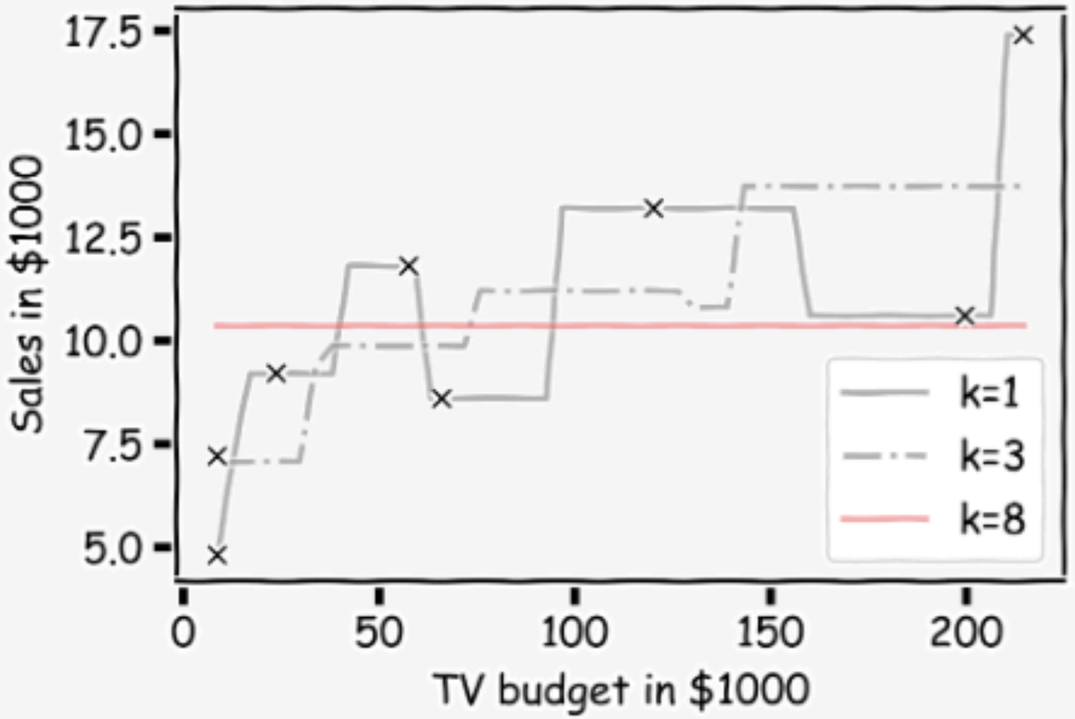
Tablular data and Pandas

DataFrame object: a two-dimensional table of data with column and row indexes. The columns are made up of pandas Series objects.

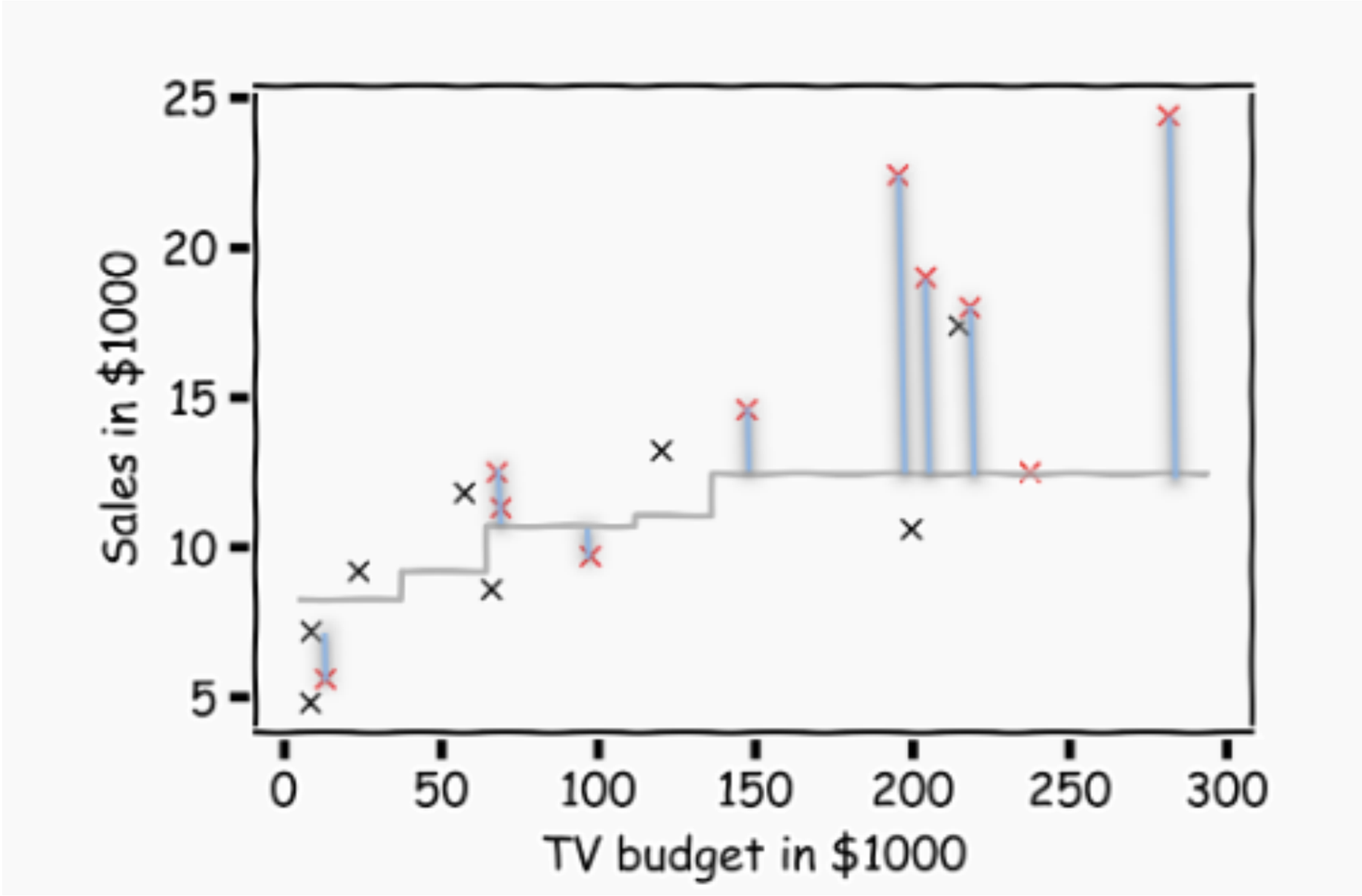


Regression

KNN Regression



Residuals and their minimization



How to fit: sklearn

```
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.2)
lr2 = LinearRegression().fit(Xtrain, ytrain)
r2_test = r2_score(ytest, lr.predict(Xtest))
r2_train = r2_score(ytrain, lr.predict(Xtrain))
```

How to fit: Keras

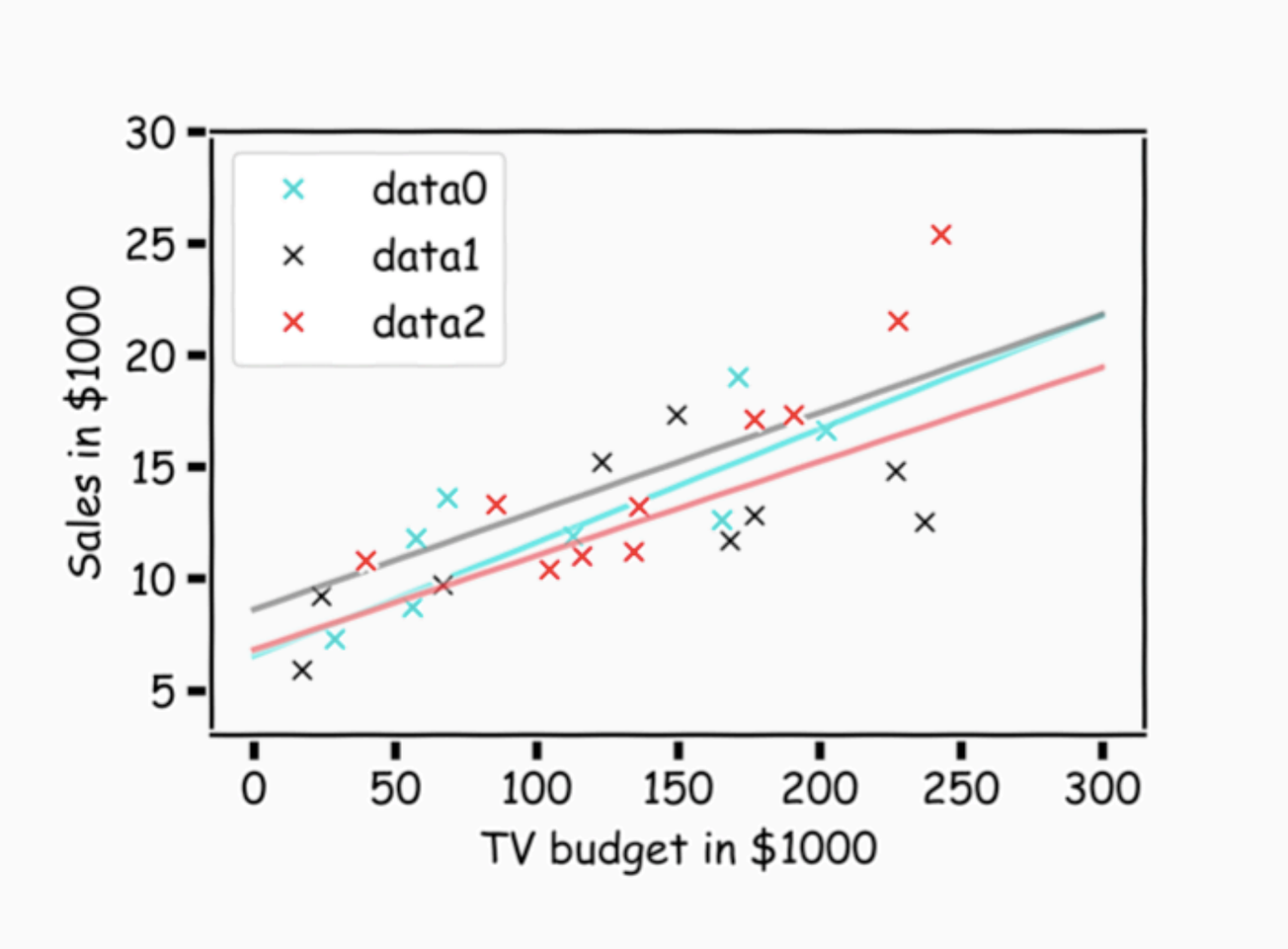
```
inputs_placeholder = Input(shape=(1,))
outputs_placeholder = Dense(1, activation='linear')(inputs_placeholder)

m = Model(inputs=inputs_placeholder, outputs=outputs_placeholder)
m.compile(optimizer='sgd', loss='mean_squared_error',  metrics=['mae','accuracy'])
m.summary()
```

Frequentist Statistics

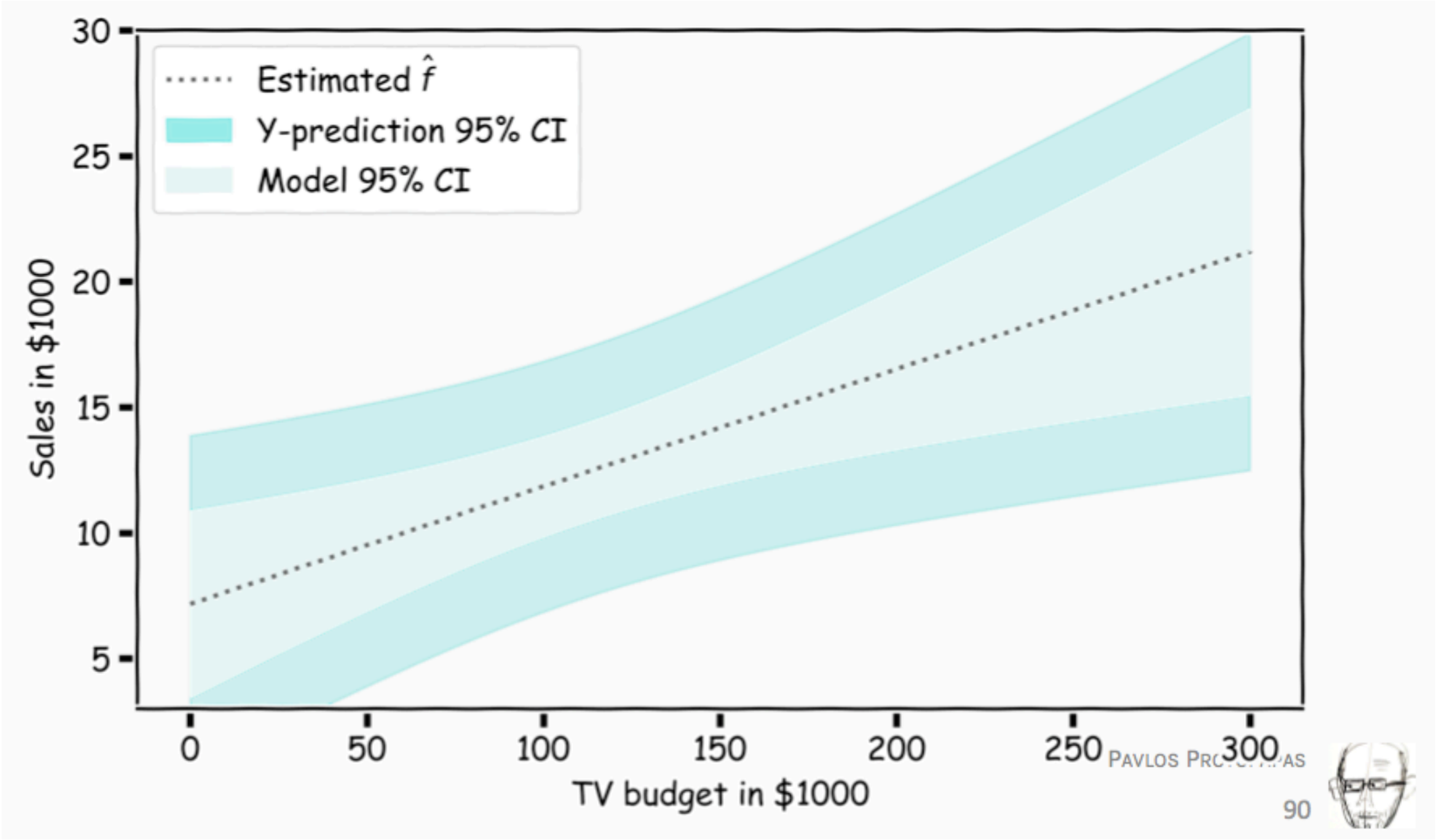
"data is a **sample** from an existing **population**"

- data is stochastic, variable; parameters fixed
- fit a parameter
- samples (or bootstrap) induce a sampling distribution on any estimator
- example of a very useful estimator: MLE

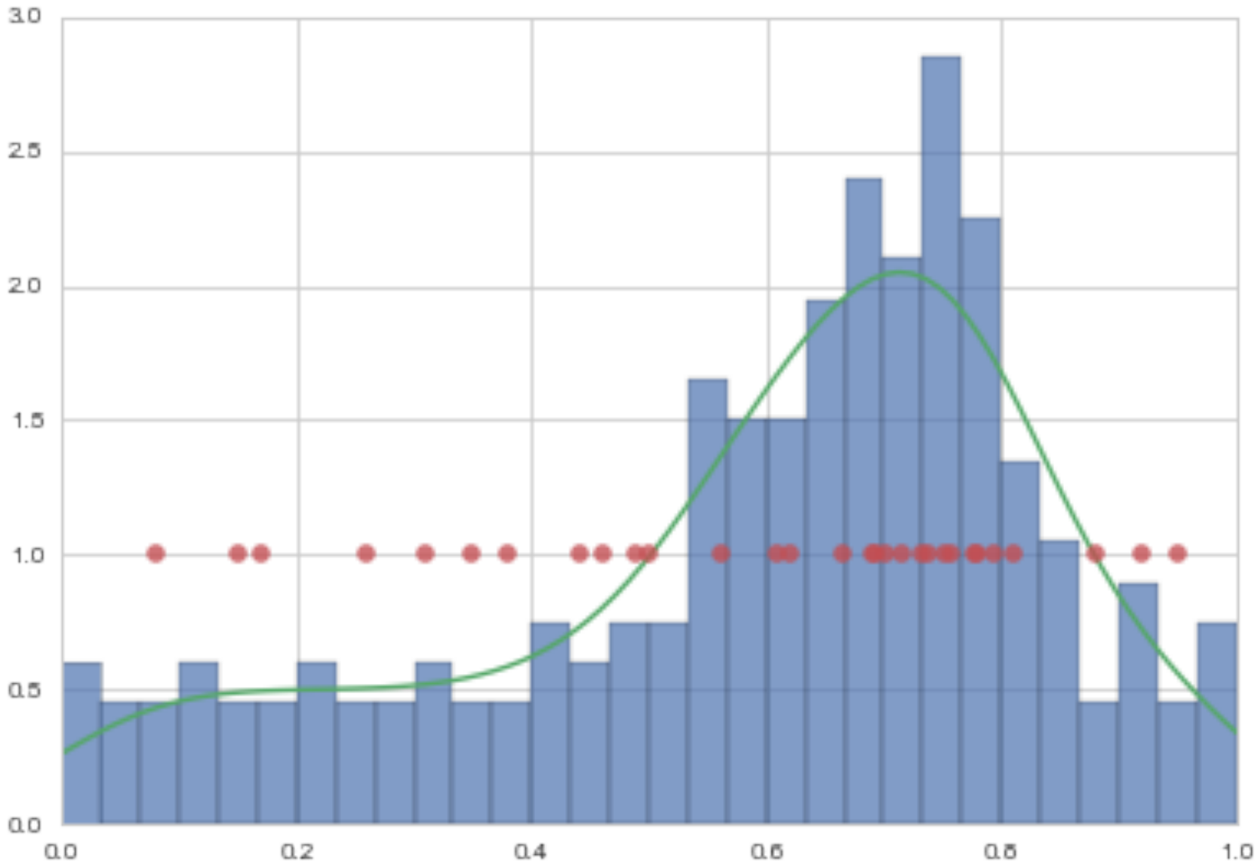
Multiple fits from multiple samples



Regression line uncertainty vs prediction uncertainty



Learning



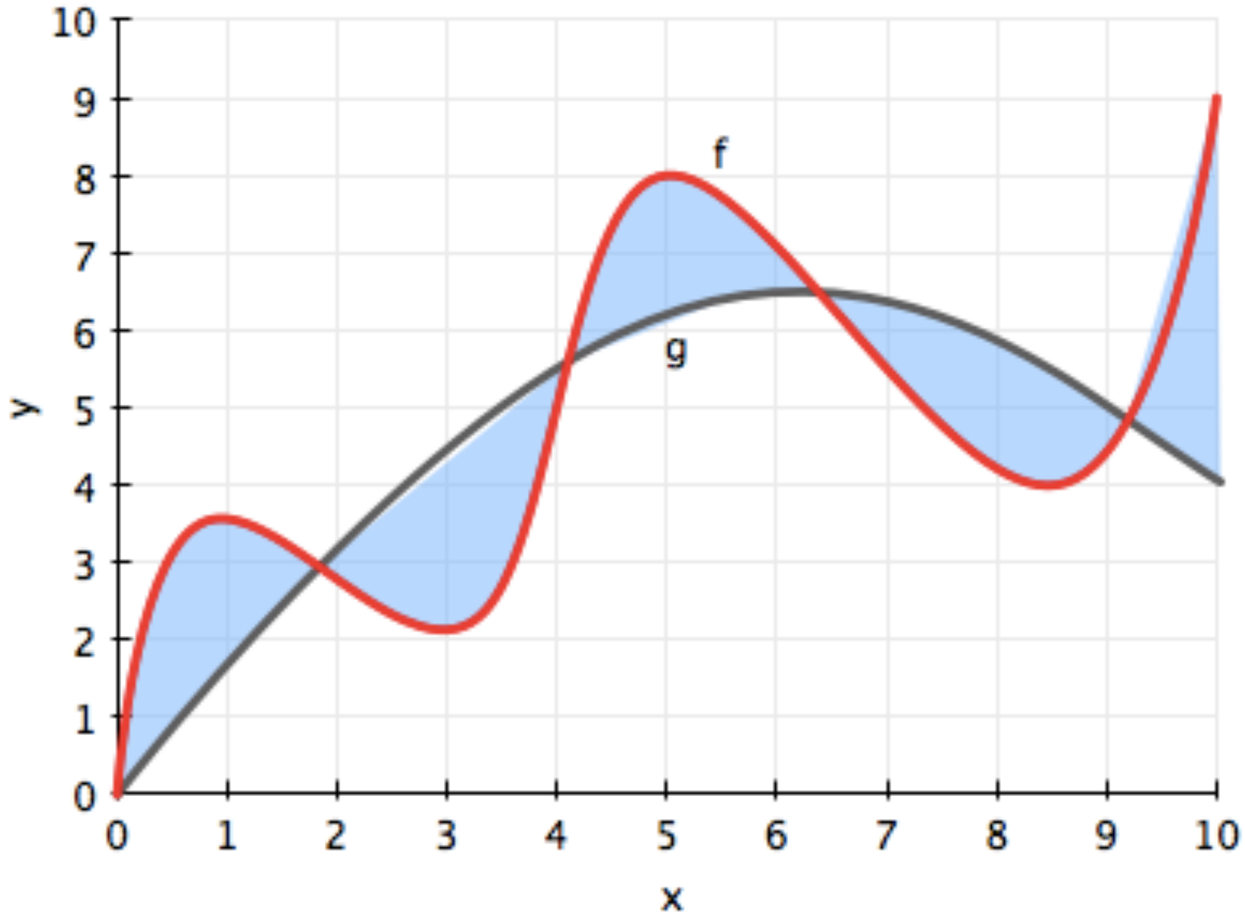
Statement of the Learning Problem

The sample must be representative of the population!

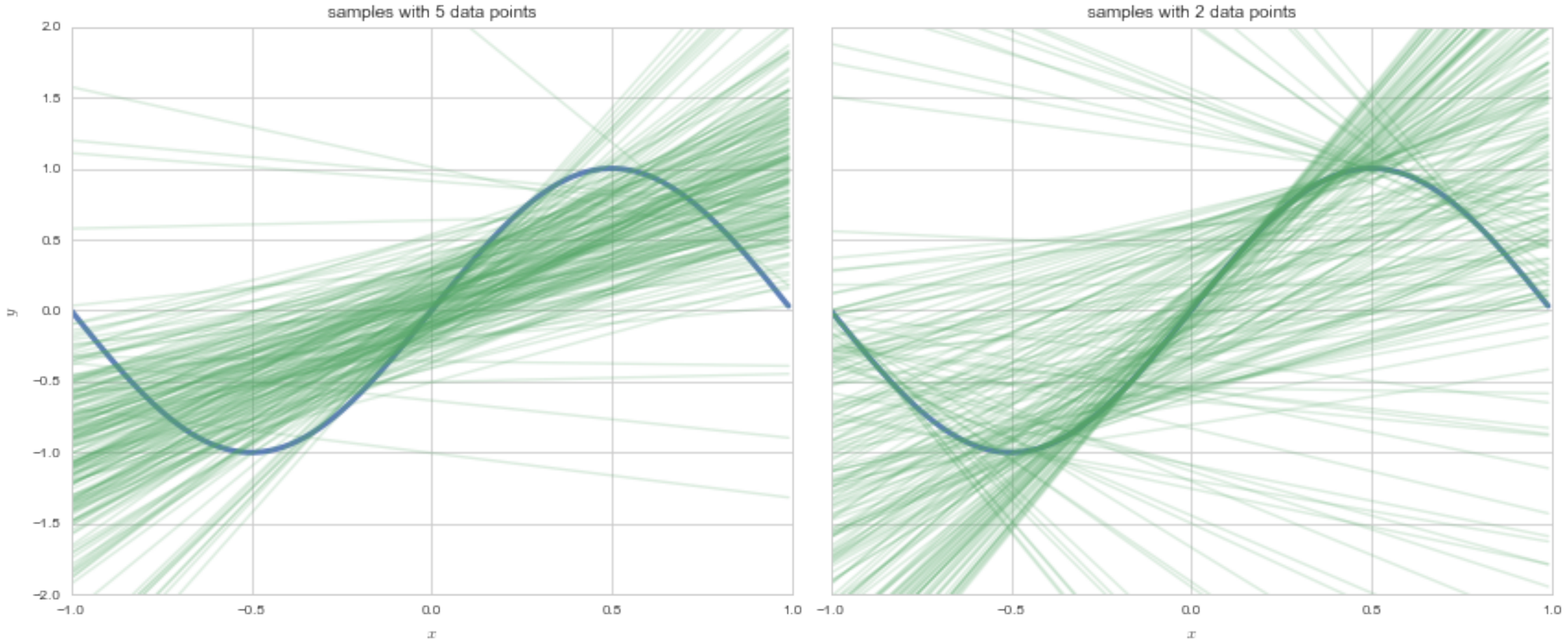
$A : R_{\mathcal{D}}(g)\ \ smallest\ on\ \mathcal{H}$
$B : R_{out}(g) \approx R_{\mathcal{D}}(g)$

A: Empirical risk estimates in-sample risk.
B: Thus the out of sample risk is also small.

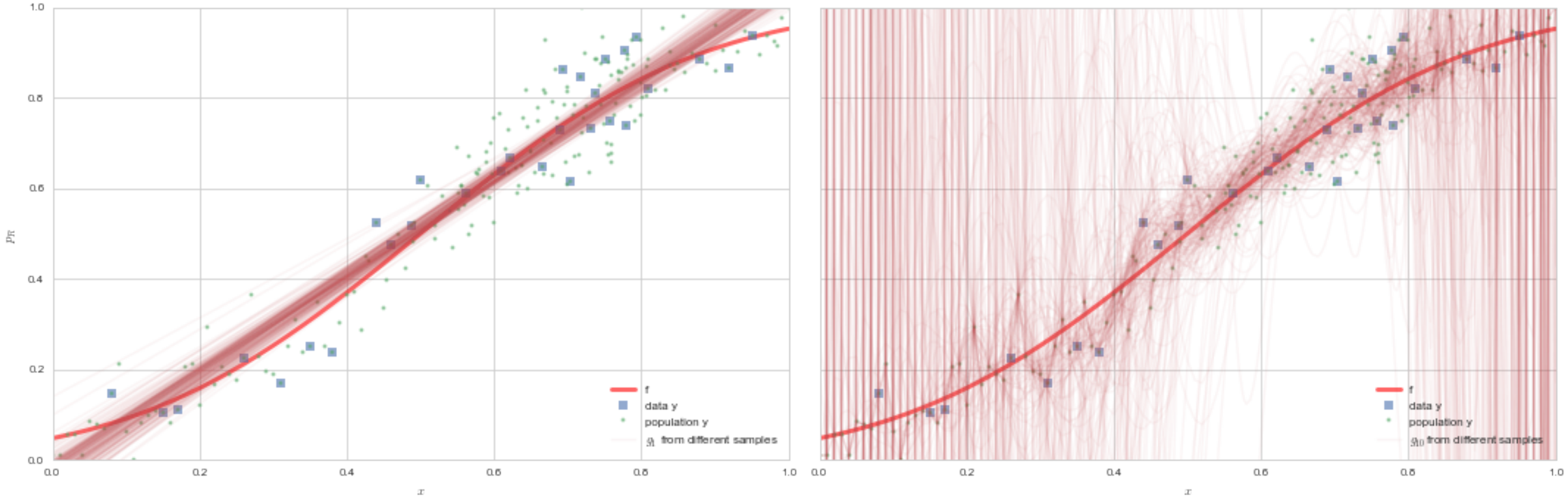
Bias or Underfitting



Data size matters

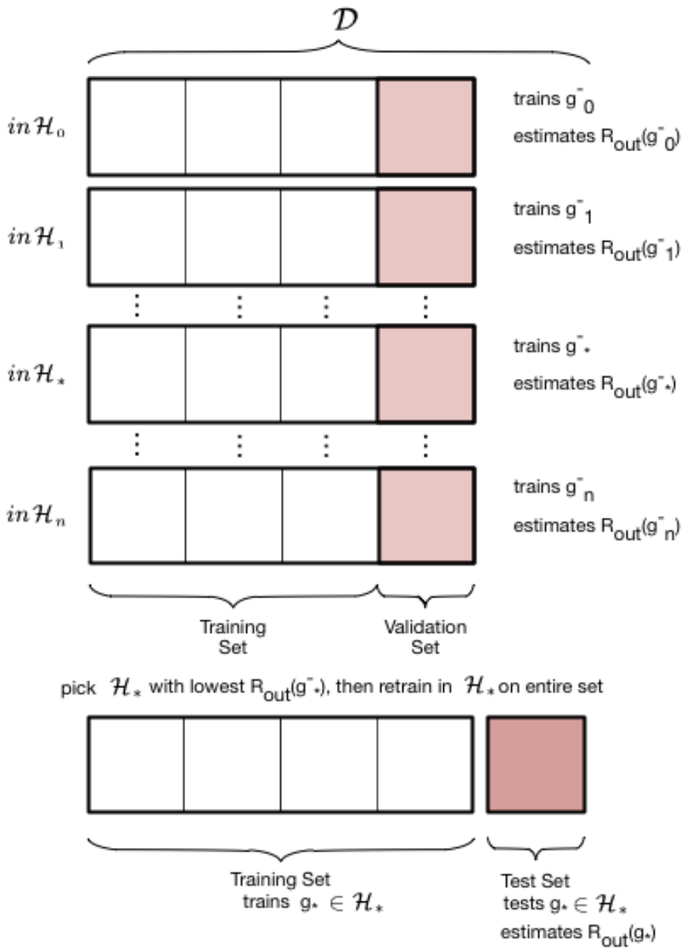


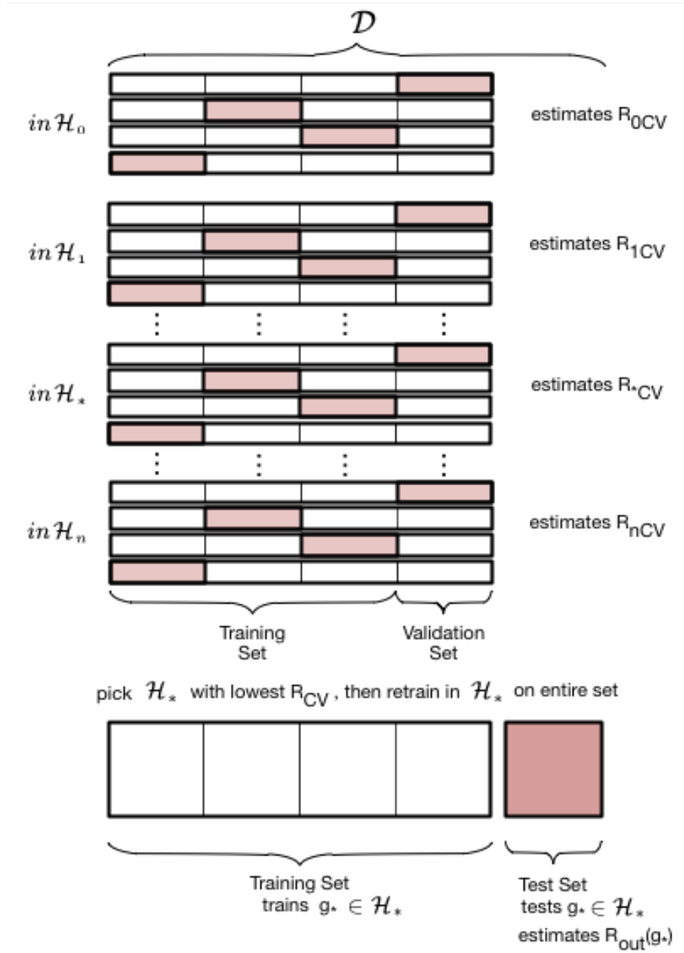
UNDERFIT(Bias) vs OVERFIT (Variance)



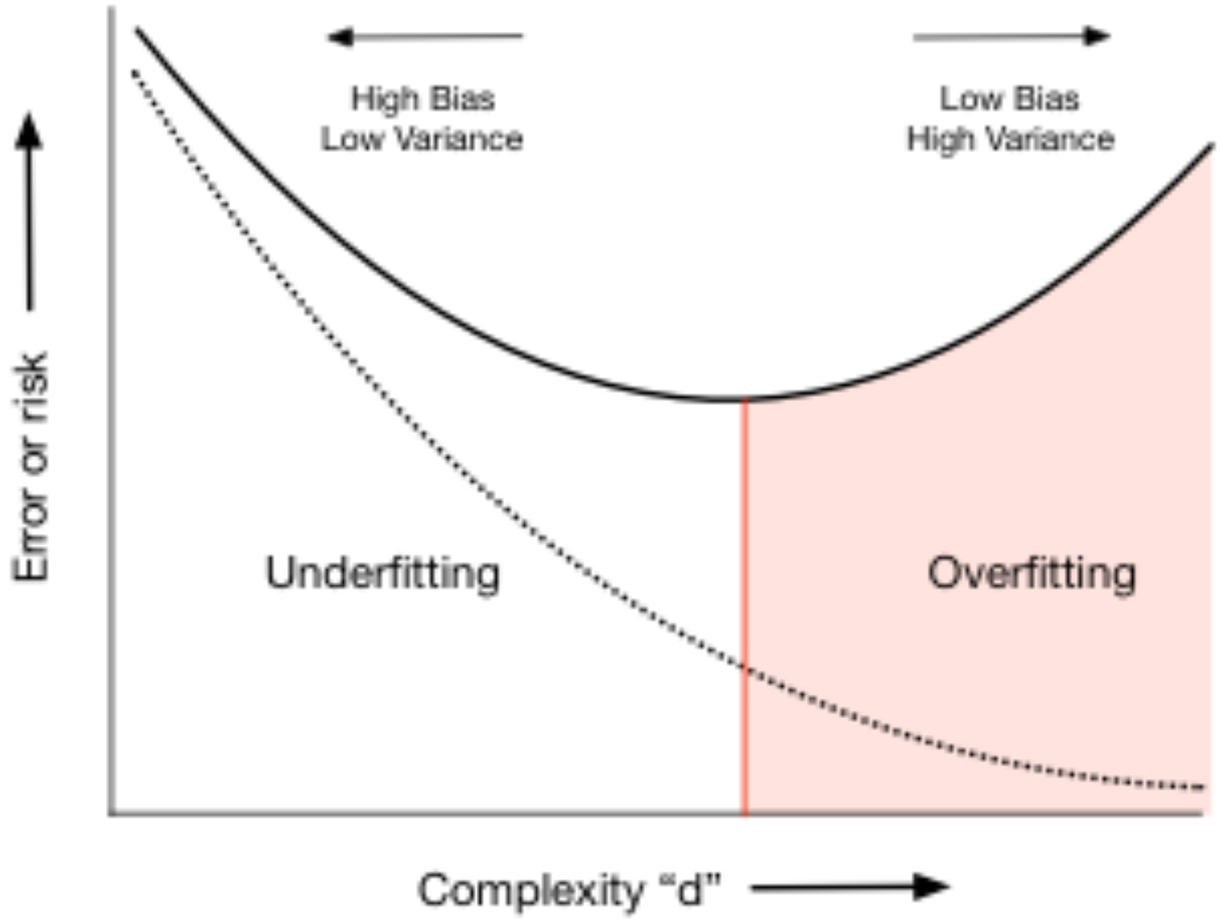

Linear regression coefficients
Value
1.25
1.00
0.75
0.50
0.25
0.00
0.75
1.00
1.25
1.50
1.75
2.00
2.25
Coefficient
1e9
Poly regression of order 10 coefficients
Value
1.5
1.0
0.5
0.0
-0.5
-1.0
-1.5
2
4
6
8
10
Coefficient

Validation, X-Val

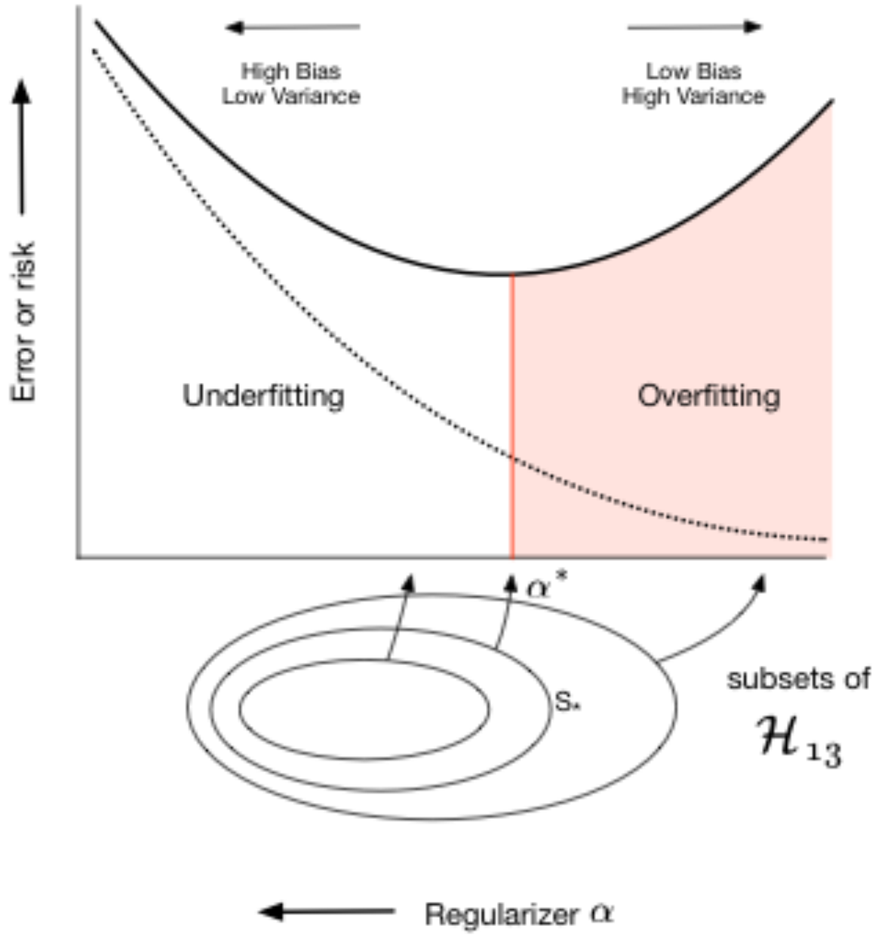




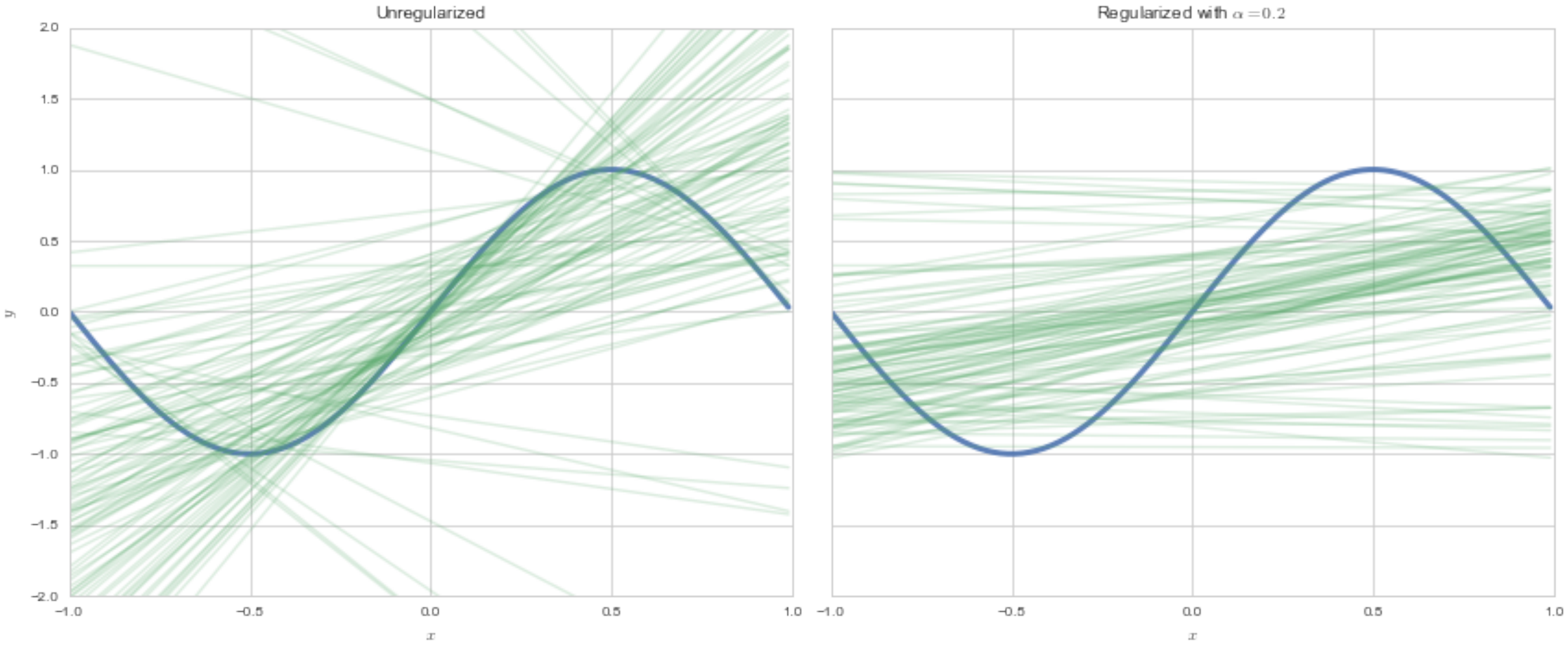
Dont Overfit



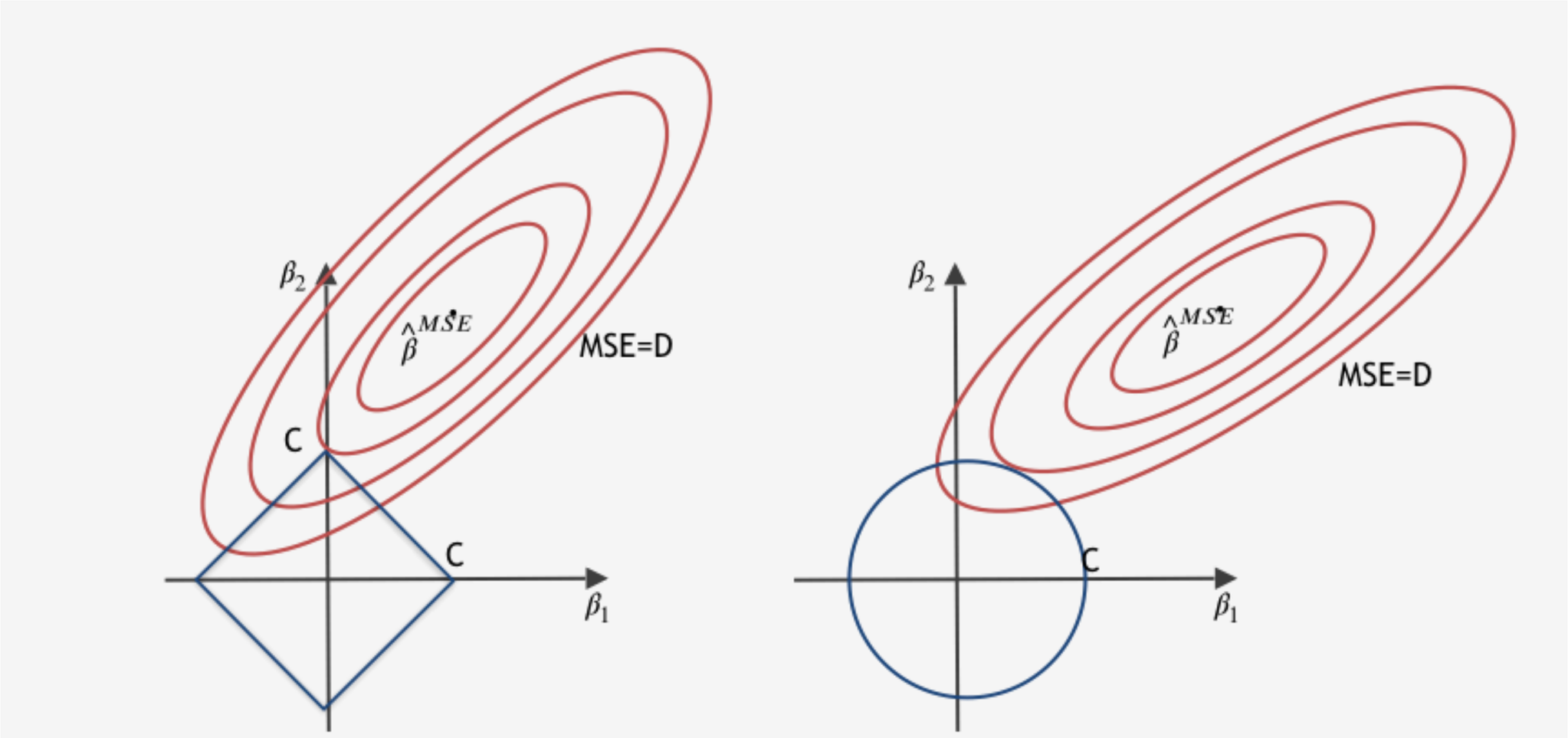
Regularization



Make em behave!

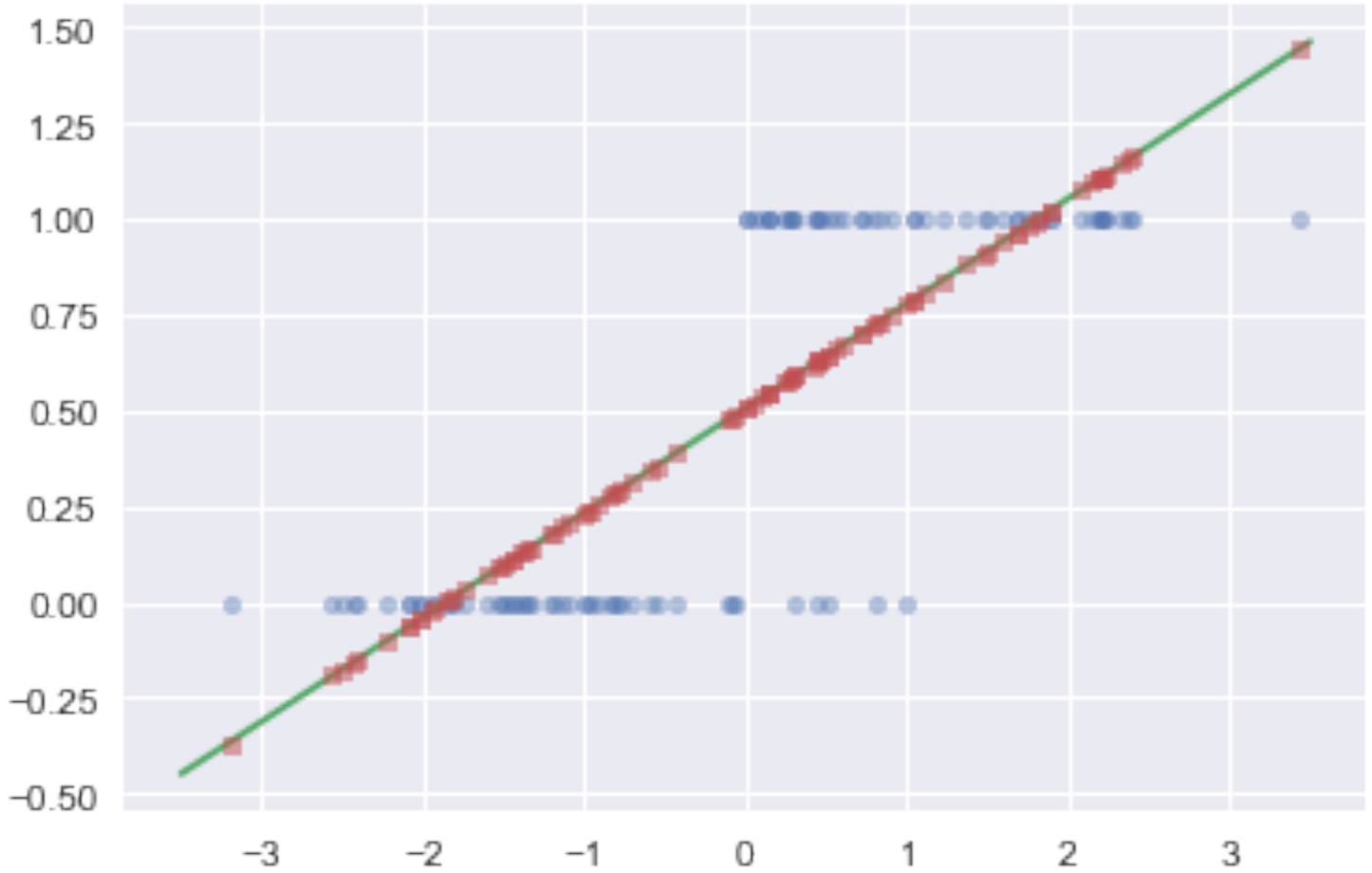


Geometry of regularization

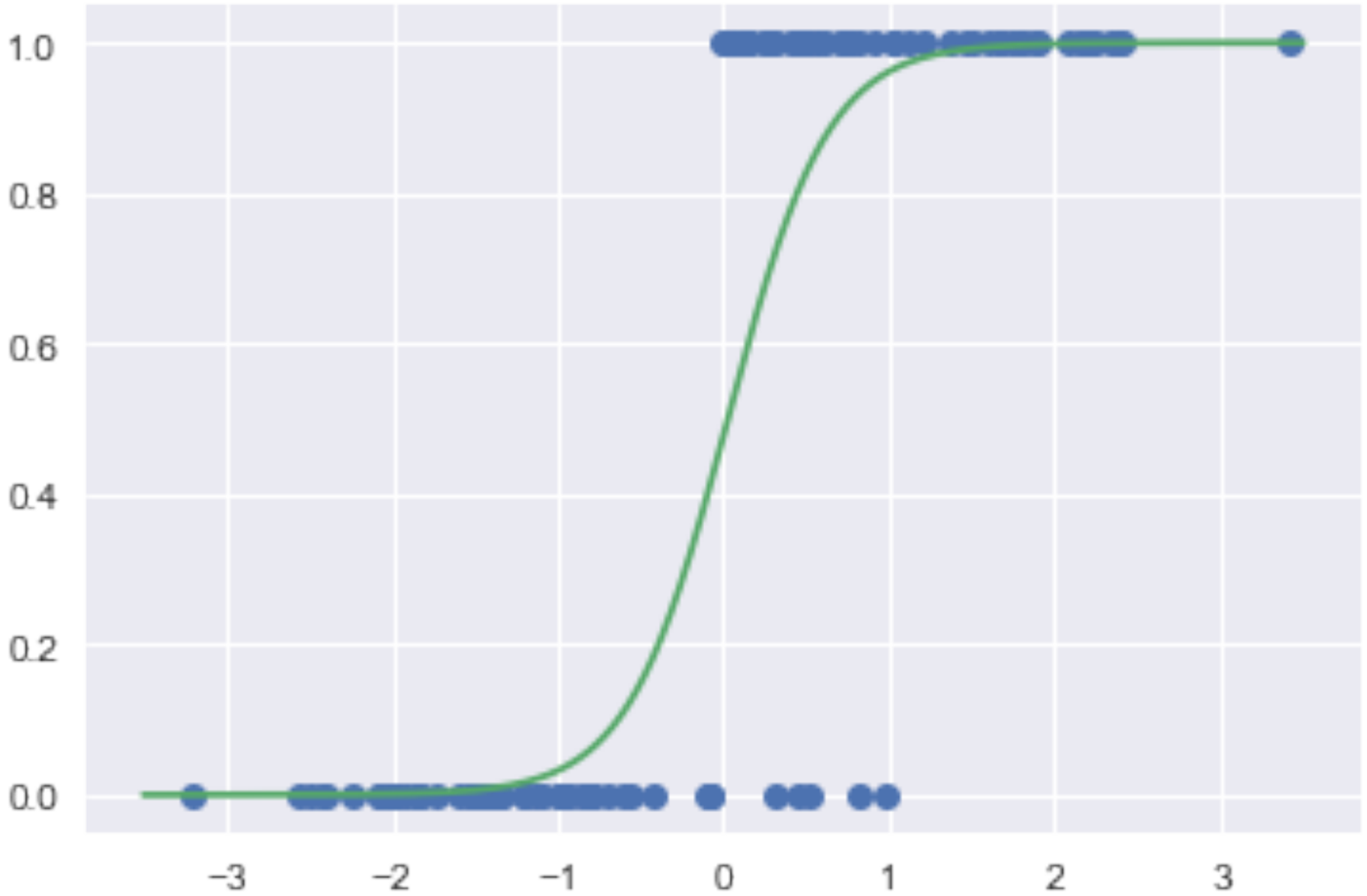


Classifcation

With Linear Regression



With Logistic Regression

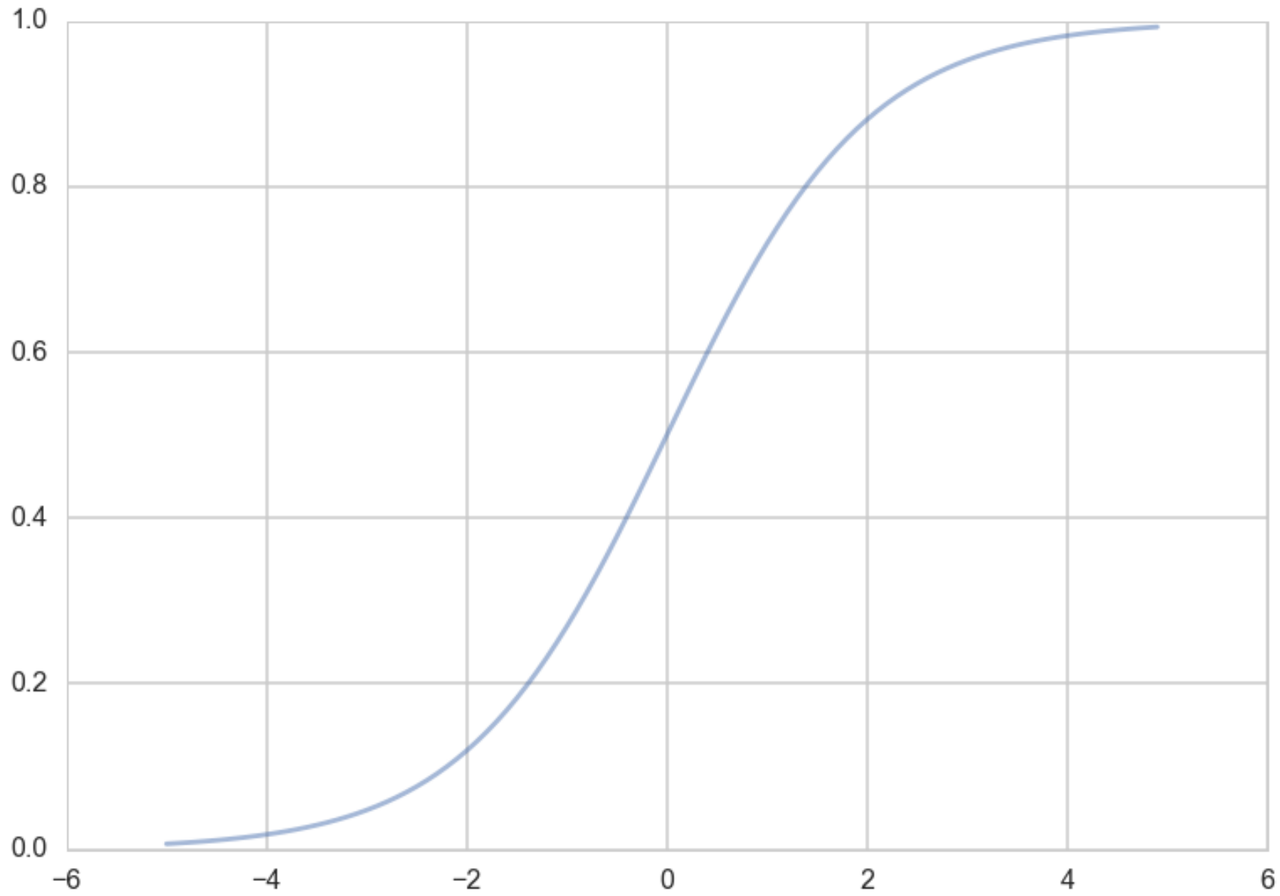


Sigmoid function

This function is plotted below:

```
h = lambda z: 1./(1+np.exp(-z))
zs=np.arange(-5,5,0.1)
plt.plot(zs, h(zs), alpha=0.5);
```

Identify: $z = \mathbf{w} \cdot \mathbf{x}$ and $h(\mathbf{w} \cdot \mathbf{x})$ with the probability that the sample is a '1' ($y = 1$).



Then, the conditional probabilities of $y = 1$ or $y = 0$ given a particular sample's features $\mathbf{x}$ are:

$$P(y = 1|\mathbf{x}) = h(\mathbf{w} \cdot \mathbf{x})$$
$$P(y = 0|\mathbf{x}) = 1 - h(\mathbf{w} \cdot \mathbf{x}).$$

These two can be written together as

$$P(y|\mathbf{x}, \mathbf{w}) = h(\mathbf{w} \cdot \mathbf{x})^y (1 - h(\mathbf{w} \cdot \mathbf{x}))^{(1-y)}$$

BERNOULLI!!

Multiplying over the samples we get:

$$P(y|\mathbf{x},\mathbf{w}) = P(\{y_i\}|\{\mathbf{x}_i\},\mathbf{w}) = \prod_{y_i \in \mathcal{D}} P(y_i|\mathbf{x}_i,\mathbf{w}) = \prod_{y_i \in \mathcal{D}} h(\mathbf{w}\cdot\mathbf{x}_i)^{y_i}\left(1-h(\mathbf{w}\cdot\mathbf{x}_i)\right)^{(1-y_i)}$$

Indeed its important to realize that a particular sample can be thought of as a draw from some "true" probability distribution.

maximum likelihood estimation maximises the **likelihood of the sample y**, or alternately the log-likelihood,

$\mathcal{L} = P(y \mid \mathbf{x}, \mathbf{w})$. OR $\ell = log(P(y \mid \mathbf{x}, \mathbf{w}))$

Thus

$$
\begin{aligned}
\ell &= log\left(\prod_{y_i \in \mathcal{D}} h(\mathbf{w}\cdot\mathbf{x}_i)^{y_i}\left(1-h(\mathbf{w}\cdot\mathbf{x}_i)\right)^{(1-y_i)}\right) \\
&= \sum_{y_i \in \mathcal{D}} log\left(h(\mathbf{w}\cdot\mathbf{x}_i)^{y_i}\left(1-h(\mathbf{w}\cdot\mathbf{x}_i)\right)^{(1-y_i)}\right) \\
&= \sum_{y_i \in \mathcal{D}} log\, h(\mathbf{w}\cdot\mathbf{x}_i)^{y_i} + log\left(1-h(\mathbf{w}\cdot\mathbf{x}_i)\right)^{(1-y_i)} \\
&= \sum_{y_i \in \mathcal{D}} (y_i log(h(\mathbf{w}\cdot\mathbf{x})) + (1-y_i) log(1-h(\mathbf{w}\cdot\mathbf{x})))
\end{aligned}
$$

Logistic Regression: NLL

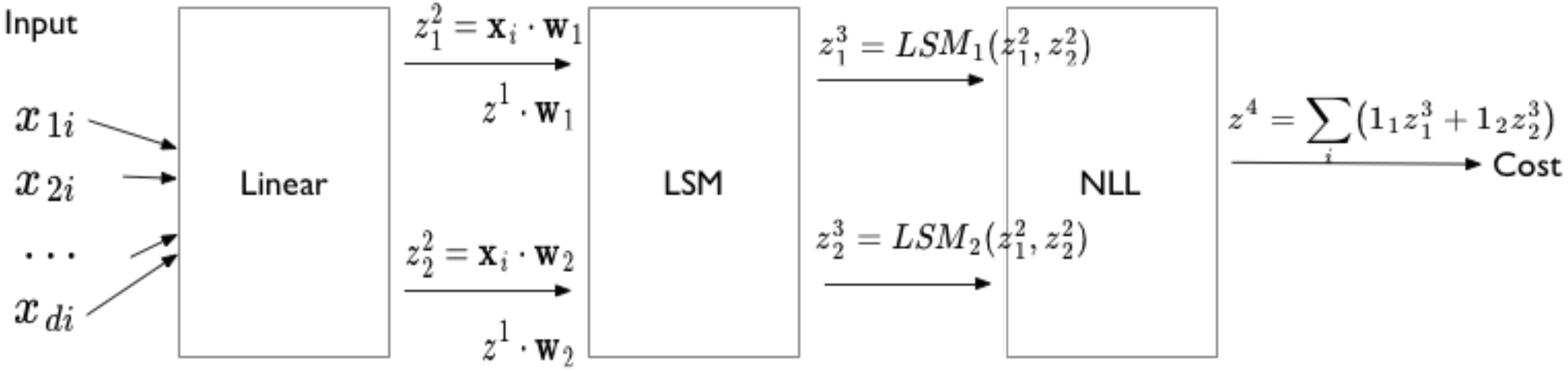
The negative of this log likelihood (NLL), also called *cross-entropy*.

$$NLL = -\sum_{y_i \in \mathcal{D}} (y_i log(h(\mathbf{w} \cdot \mathbf{x})) + (1 - y_i) log(1 - h(\mathbf{w} \cdot \mathbf{x})))$$

Gradient: $\nabla_{\mathbf{w}} NLL = \sum_i \mathbf{x}_i^T (p_i - y_i) = \mathbf{X}^T \cdot (\mathbf{p} - \mathbf{w})$

Hessian: $H = \mathbf{X}^T diag(p_i(1 - p_i))\mathbf{X}$ positive definite $\implies$ convex

Softmax Formulation of Logistic Regression



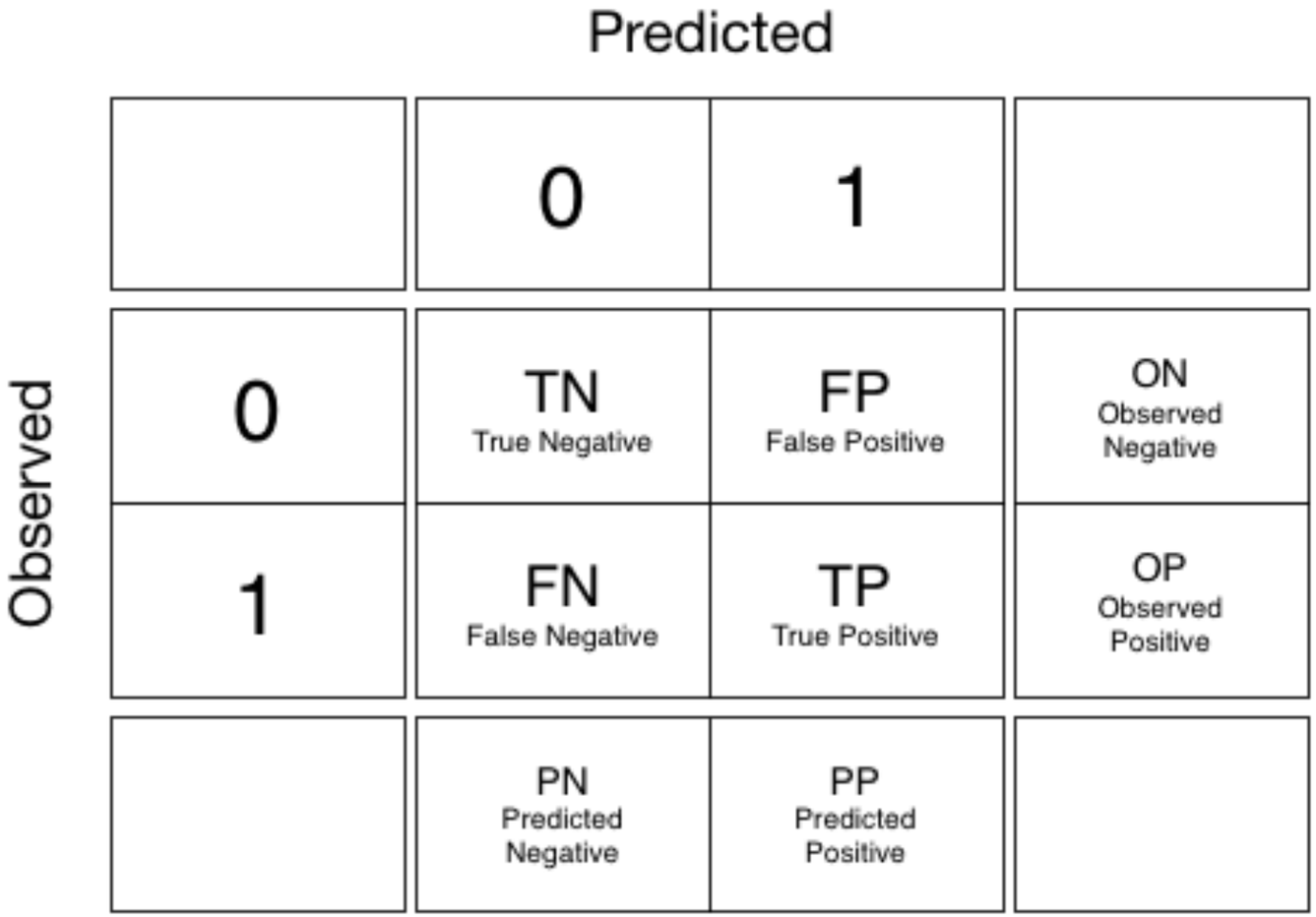
$z^1 = \mathbf{x}_i$

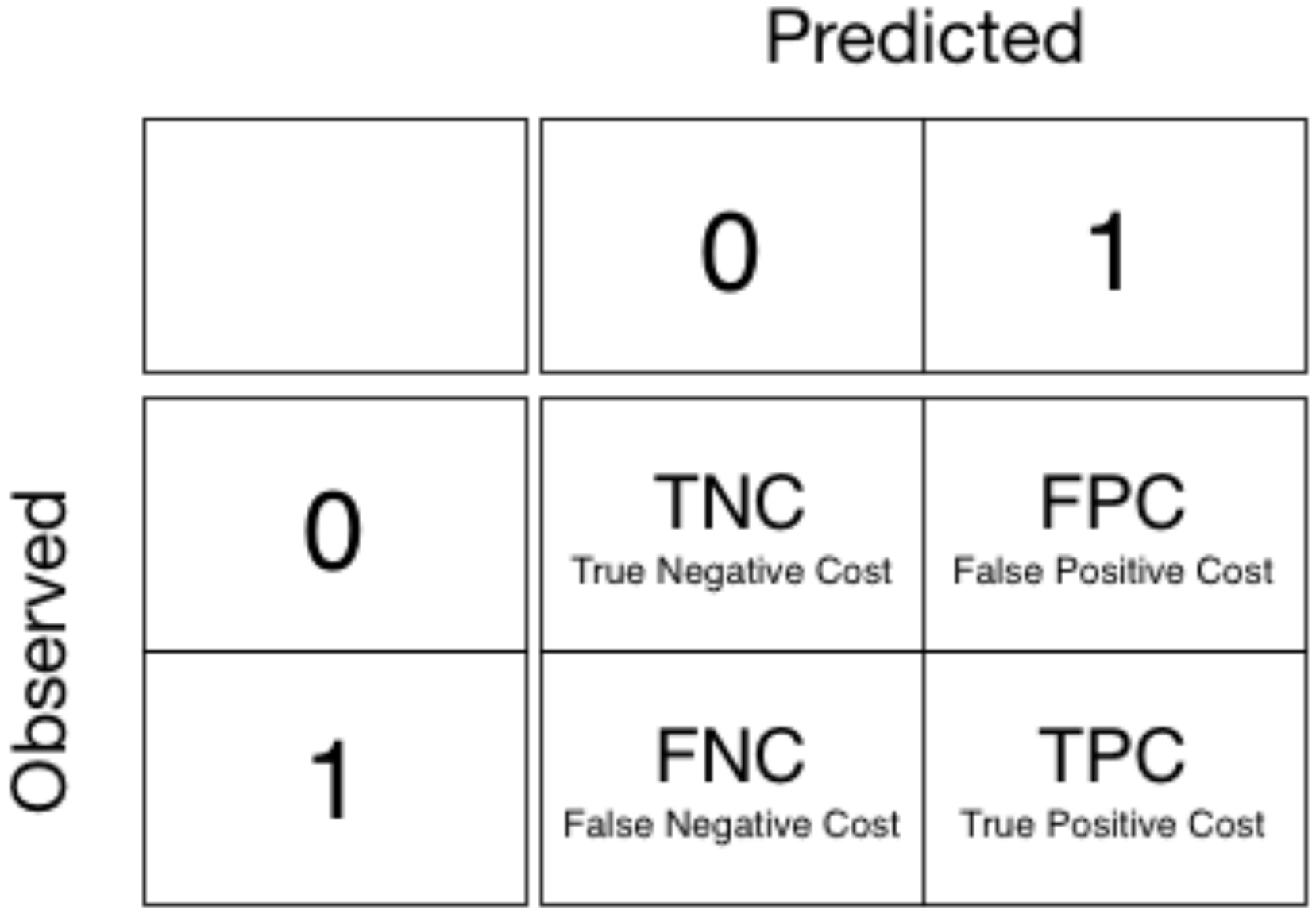
```
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# one hot encode outputs
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
# create model
model = Sequential()
model.add(Flatten(input_shape=(img_width, img_height)))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam',
              metrics=['accuracy'])
model.summary()
# Fit the model
model.fit(X_train, y_train, epochs=10, validation_data=(X_test, y_test),
          callbacks=[WandbCallback(data_type="image", labels=labels, save_model=False)])
```

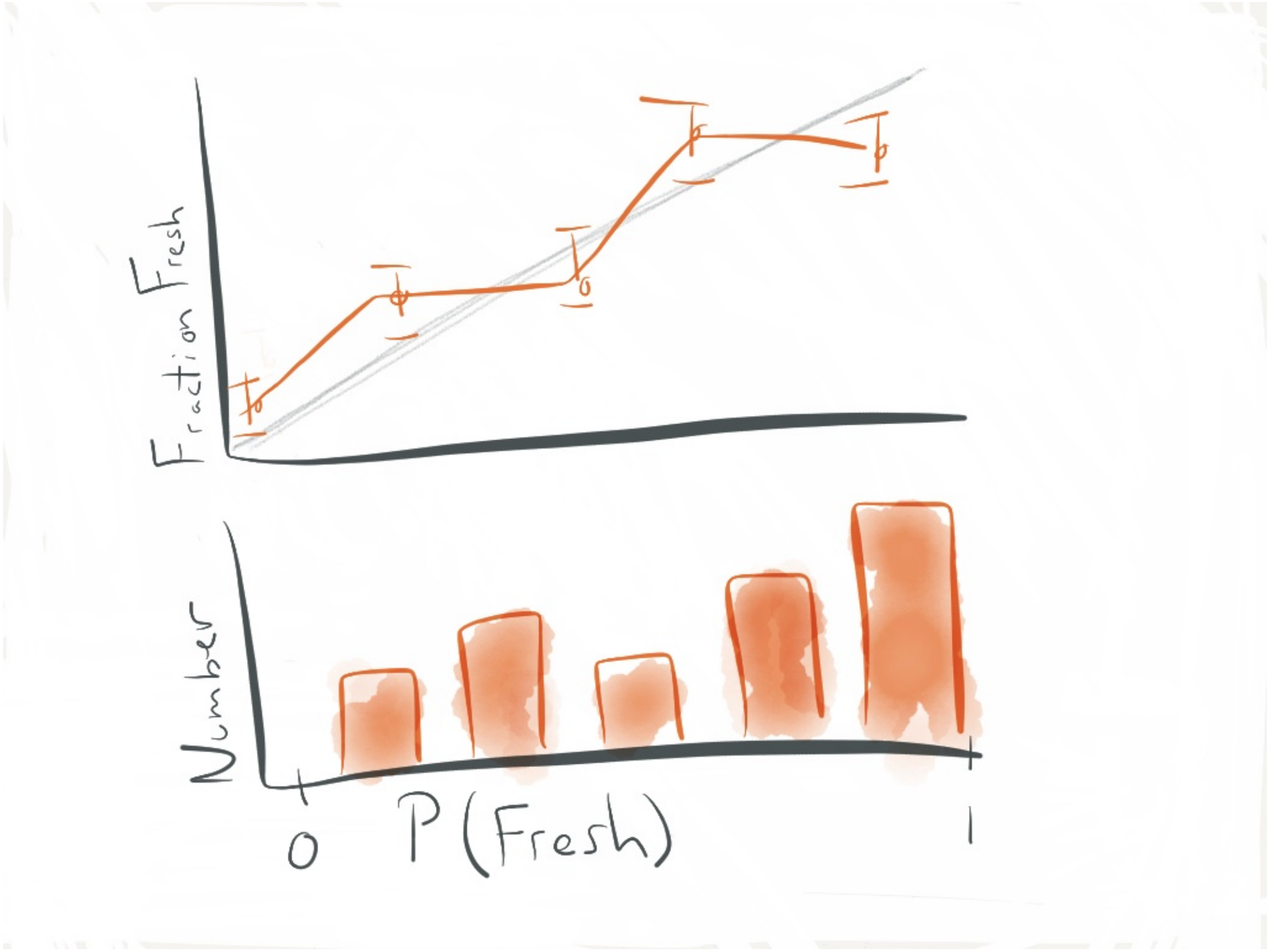
```
from sklearn.model_selection import GridSearchCV

pipeline = make_pipeline(TfidfVectorizer(),
                         LogisticRegression())

grid = GridSearchCV(pipeline,
                    param_grid={'logisticregression__C': [.1, 1, 10, 100]}, cv=5)

grid.fit(text_train, y_train)
print("Score",grid.score(text_test, y_test))
```

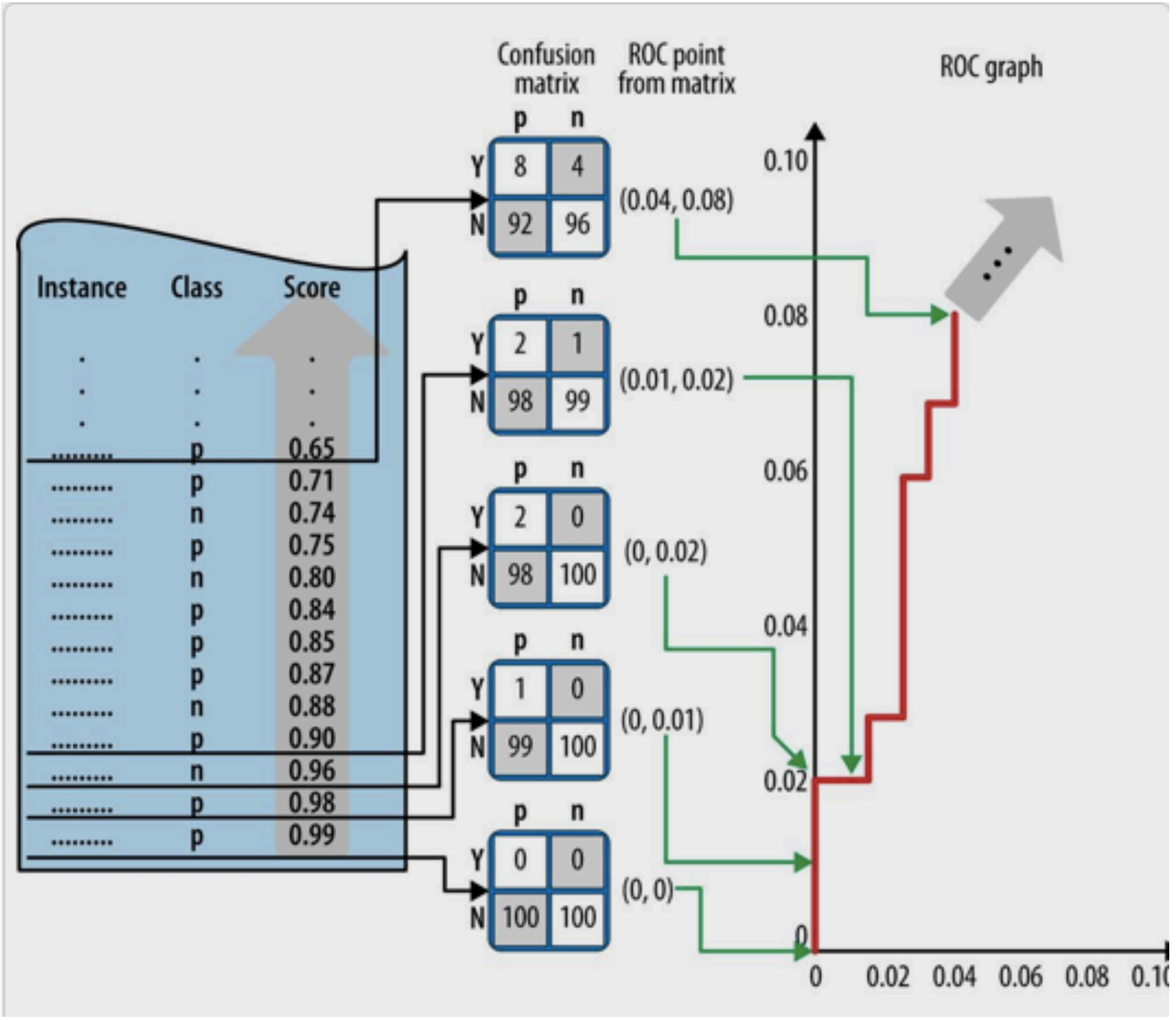
Metrics and Decision Theory

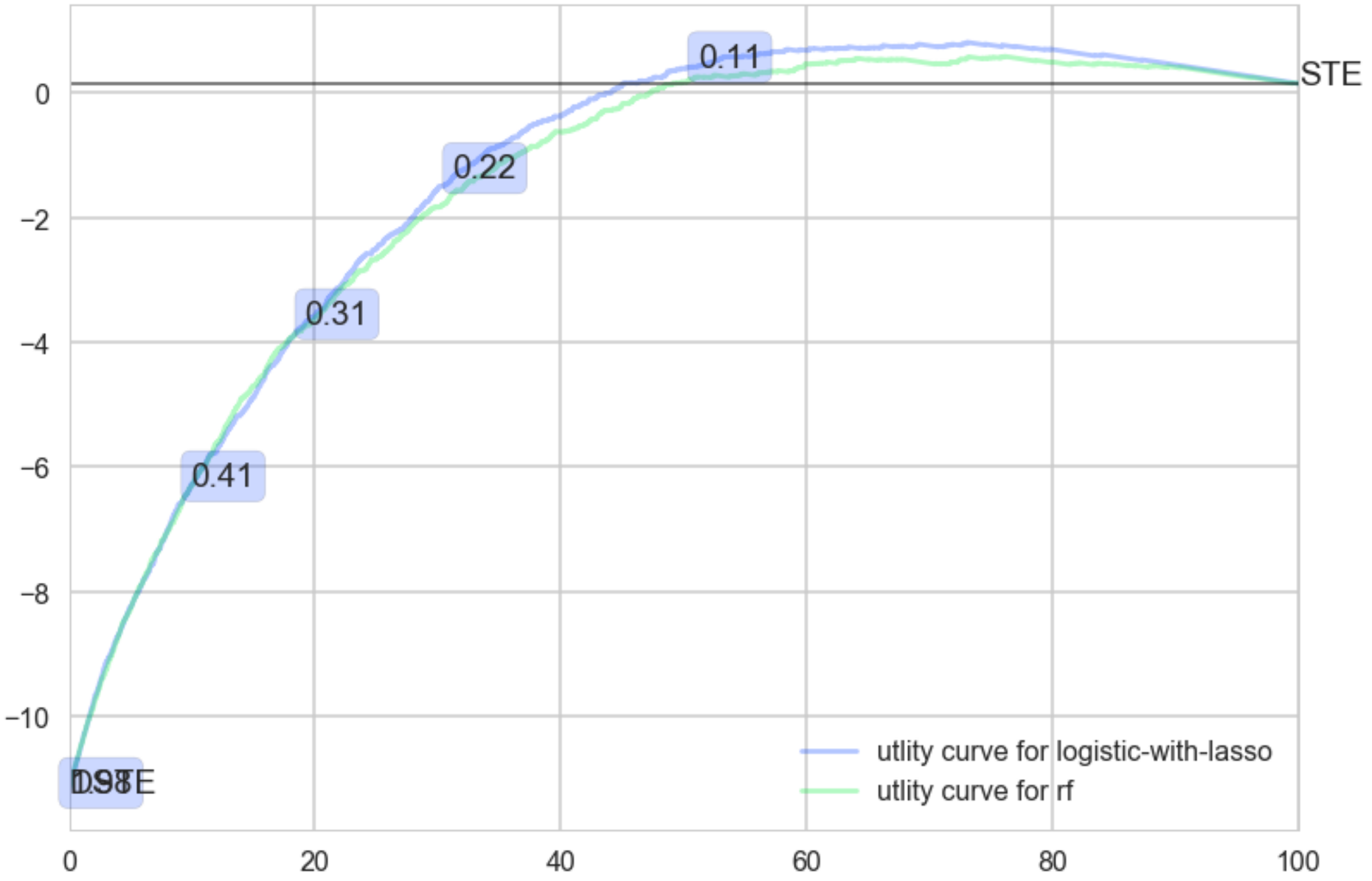
Confusion Matrix



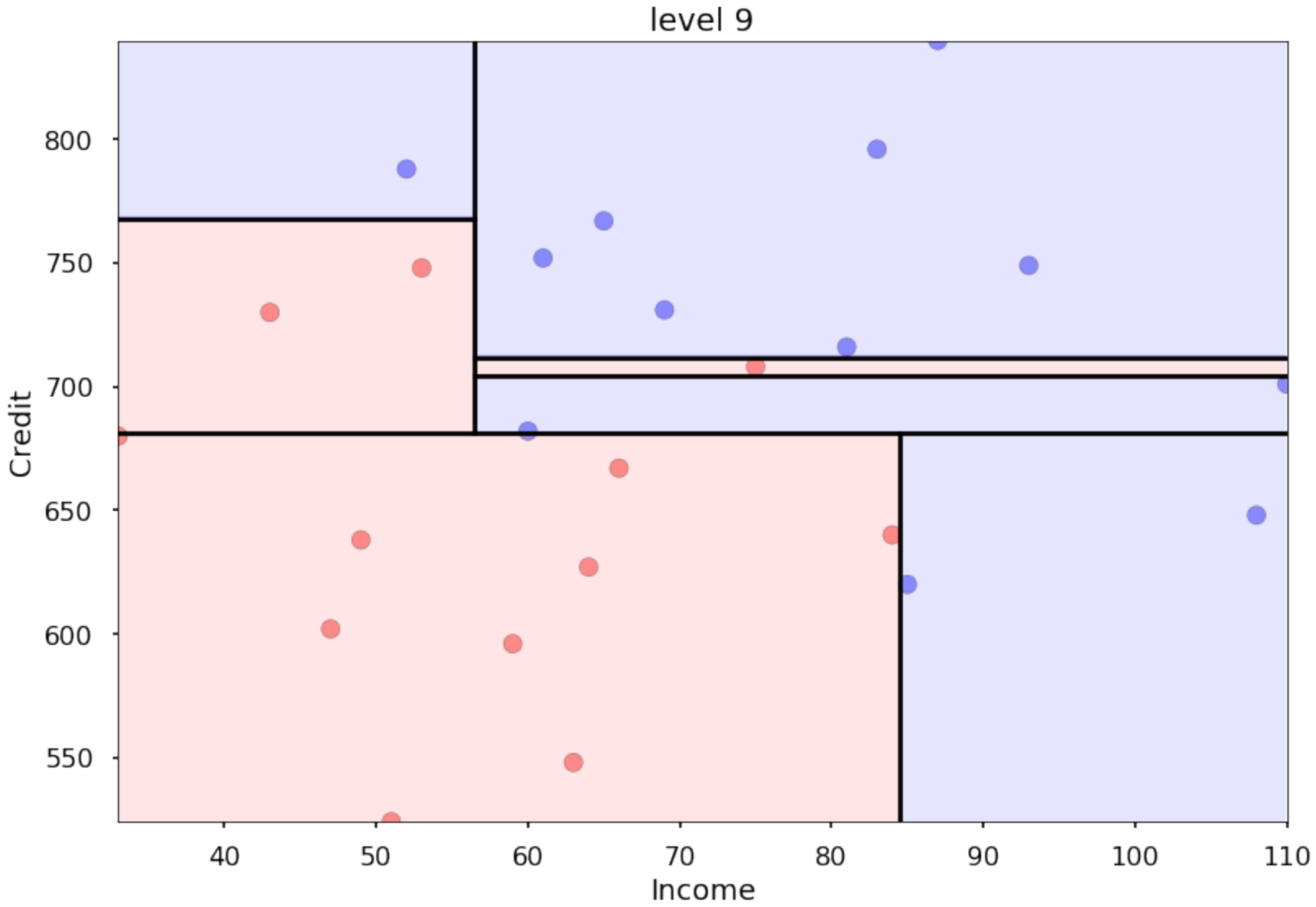

Predicted
0
1
Observed
0
TNC
True Negative Cost
FPC
False Positive Cost
1
FNC
False Negative Cost
TPC
True Positive Cost

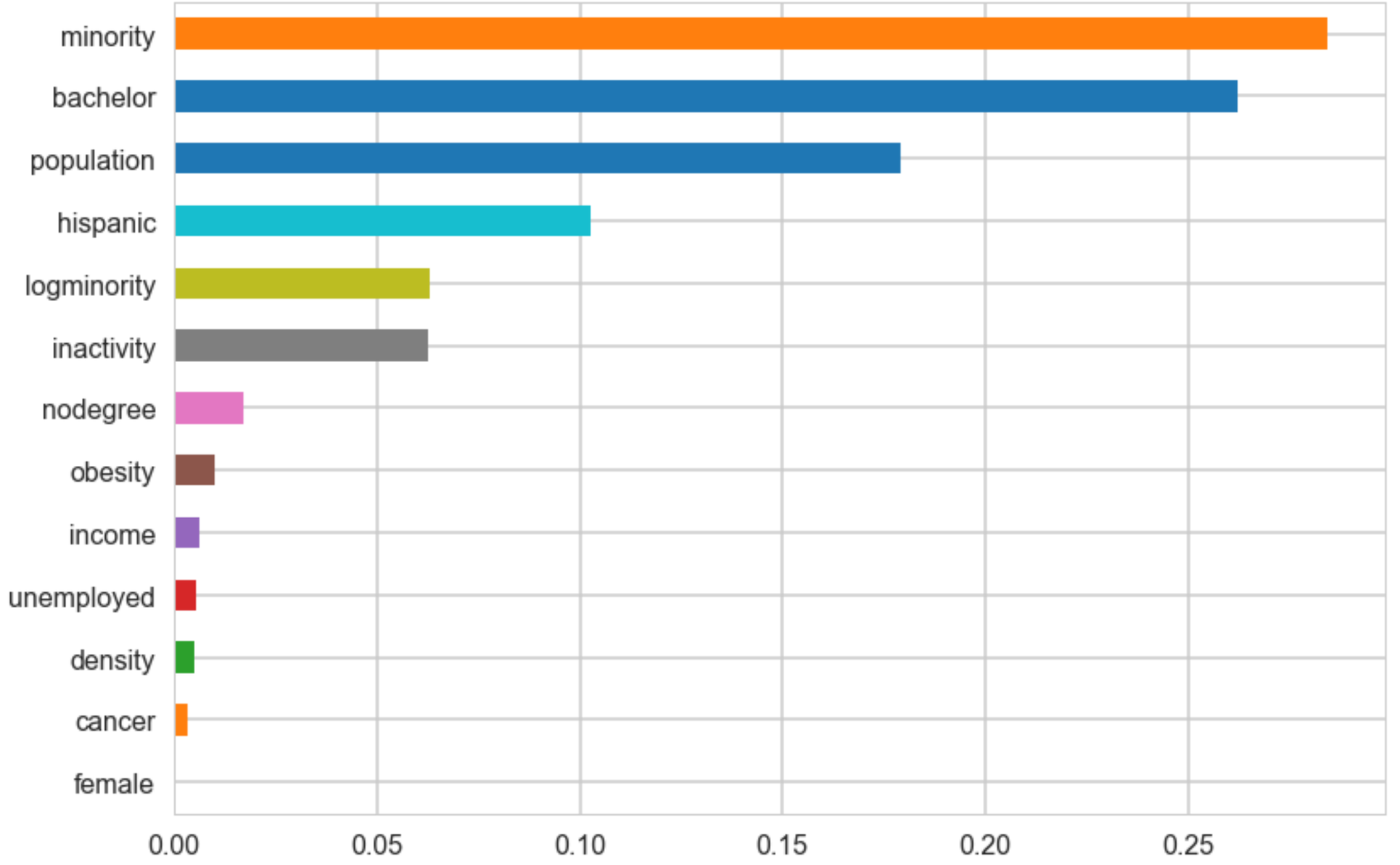

Fraction Fresh
Number
0
P(Fresh)
1

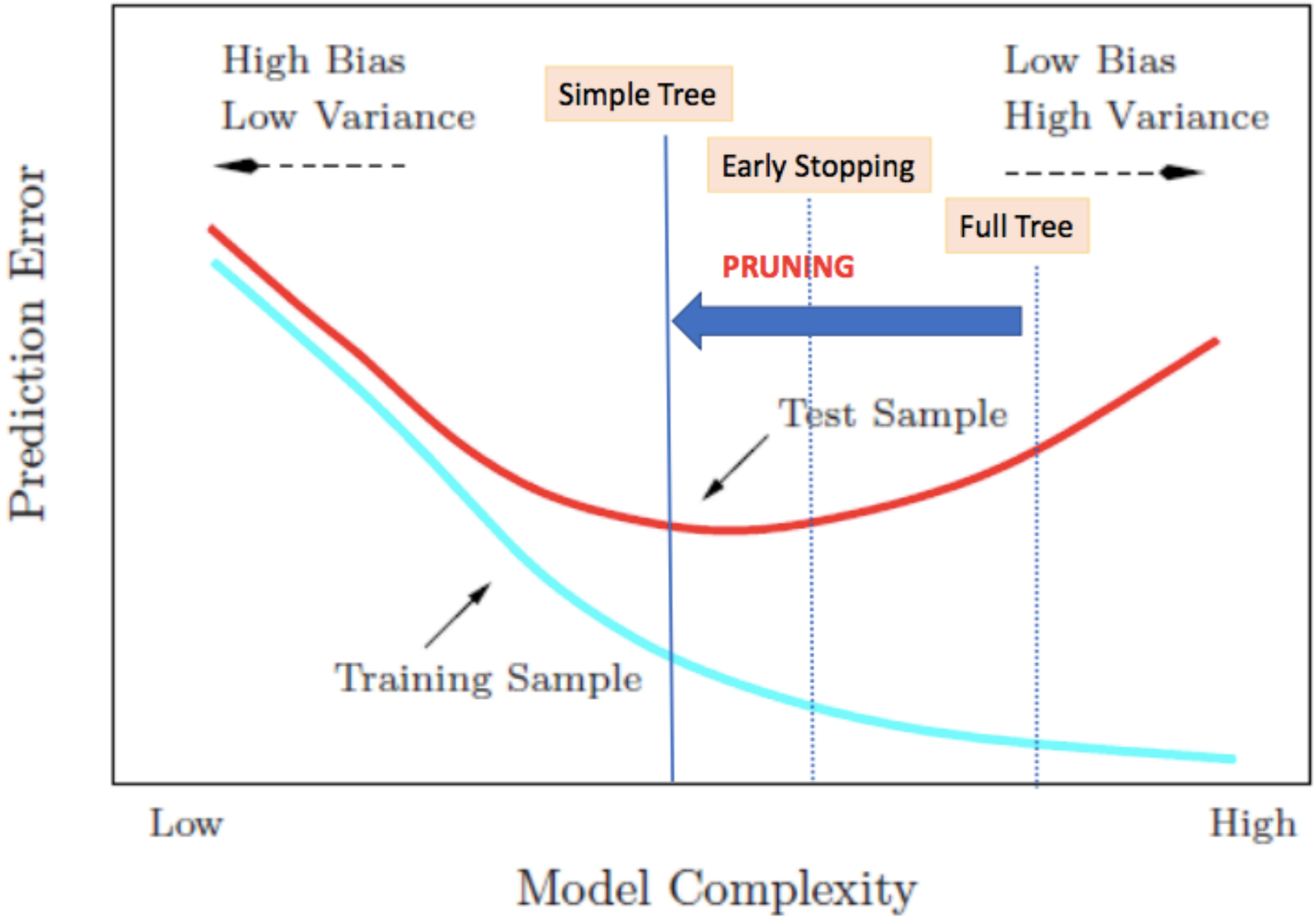

Confusion matrix
ROC point from matrix
ROC graph
Instance
Class
Score
p
n
Y
N
8
4
92
96
(0.04, 0.08)
2
1
98
99
(0.01, 0.02)
2
0
98
100
(0, 0.02)
1
0
99
100
(0, 0.01)
0
0
100
100
(0, 0)
0.65
0.71
0.74
0.75
0.80
0.84
0.85
0.87
0.88
0.90
0.96
0.98
0.99
0.10
0.08
0.06
0.04
0.02
0
0.02
0.04
0.06
0.08

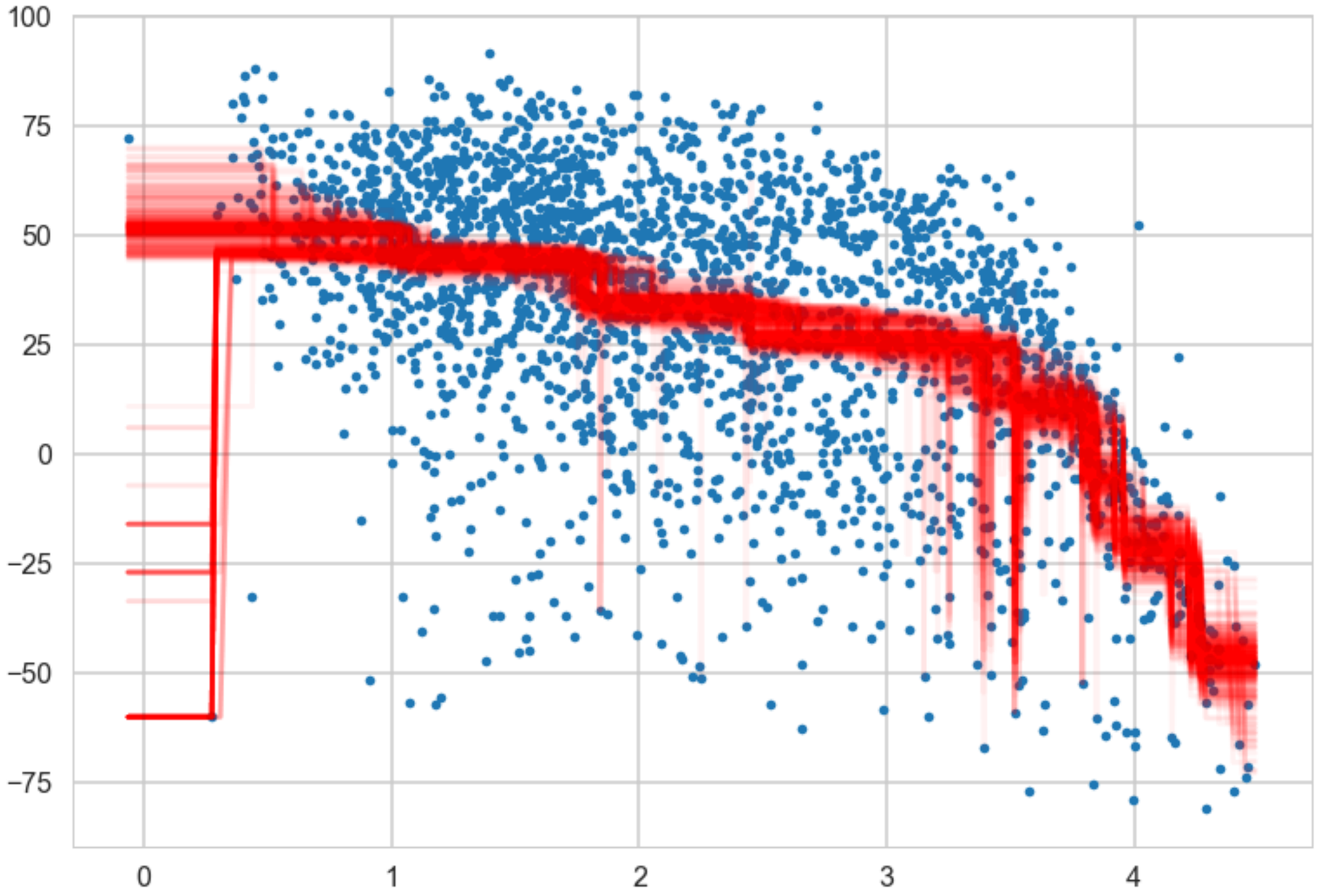

0.11
STE
0.22
0.31
0.41
0
−2
−4
−6
−8
−10
0
20
40
60
80
100
utlity curve for logistic-with-lasso
utlity curve for rf

Trees


level 9
800
750
700
650
600
550
Credit
40
50
60
70
80
90
100
110
Income


minority
bachelor
population
hispanic
logminority
inactivity
nodegree
obesity
income
unemployed
density
cancer
female
0.00
0.05
0.10
0.15
0.20
0.25


High Bias
Low Variance
Simple Tree
Early Stopping
Full Tree
Low Bias
High Variance
PRUNING
Test Sample
Training Sample
Prediction Error
Low
High
Model Complexity


100
75
50
25
0
−25
−50
−75
0
1
2
3
4

RF

- first do bagging
- then randomly choose features
- Random forest, squeezes out variance
- Big idea is ensembling

Boosting

- use weak learners
- fit residuals
- gradient descent in "data" space
- inspiration for resnets

Neural Networks

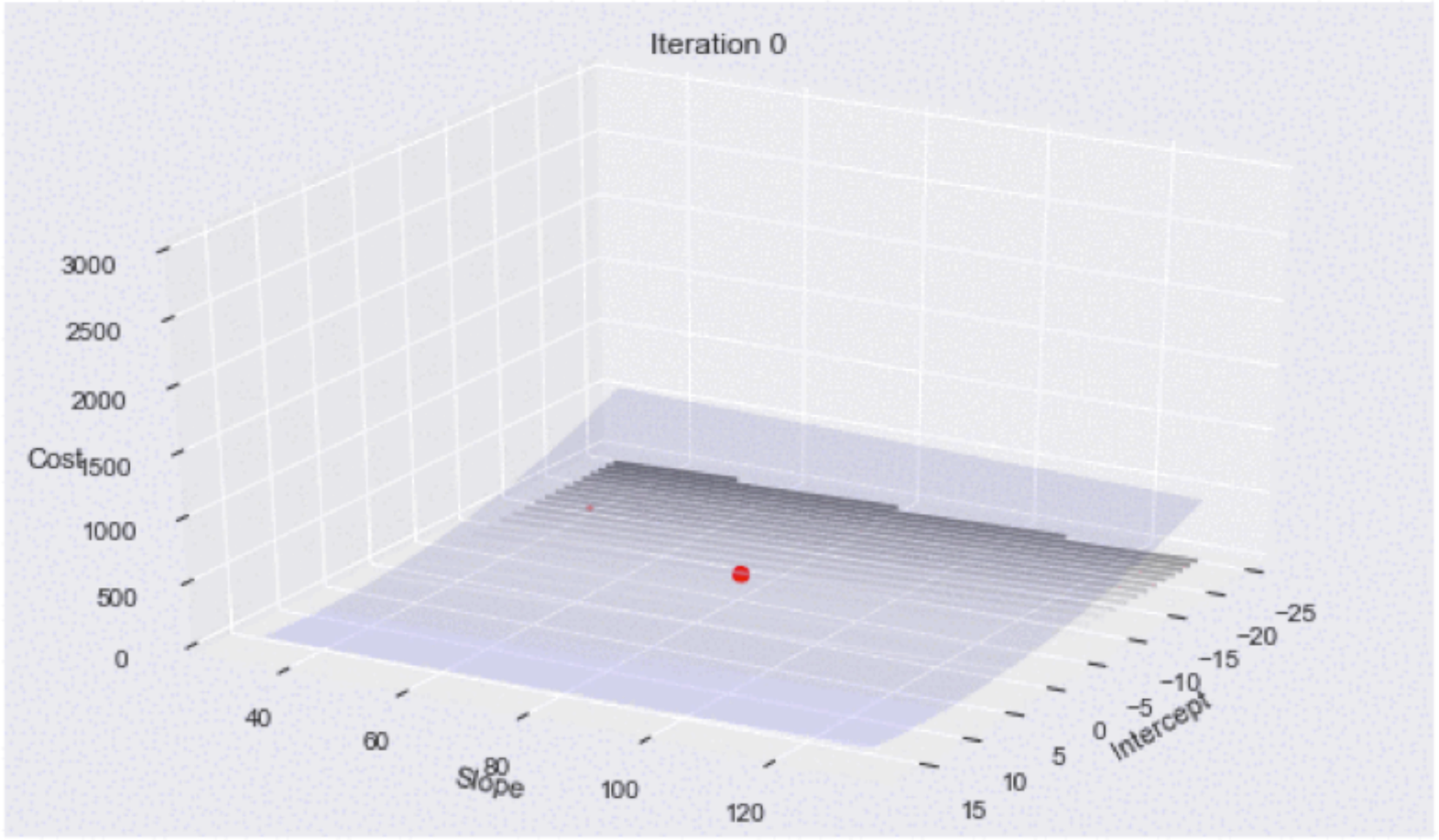
Stochastic Gradient Descent

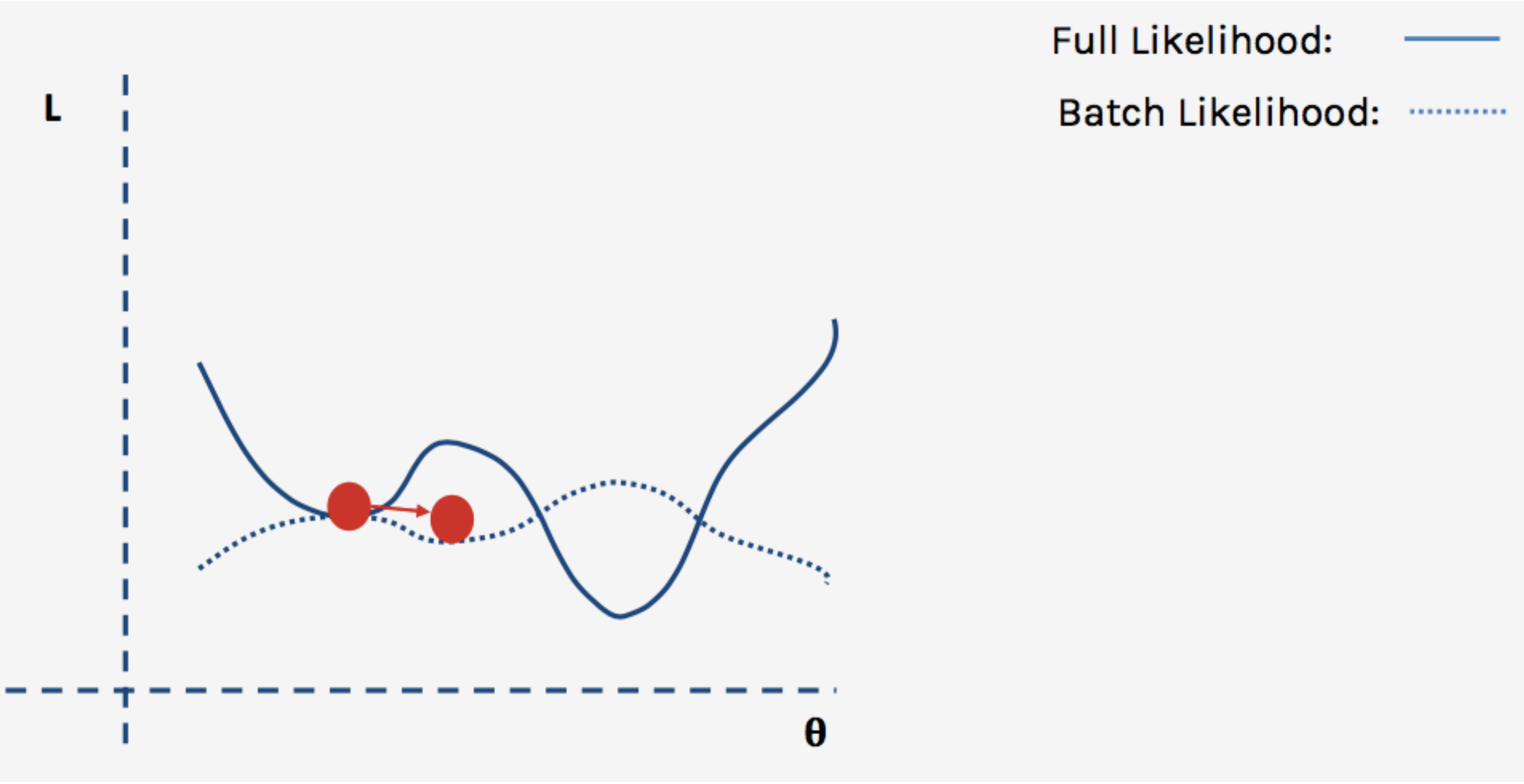
$$\theta := \theta - \alpha \nabla_\theta J_i(\theta)$$

ONE POINT AT A TIME

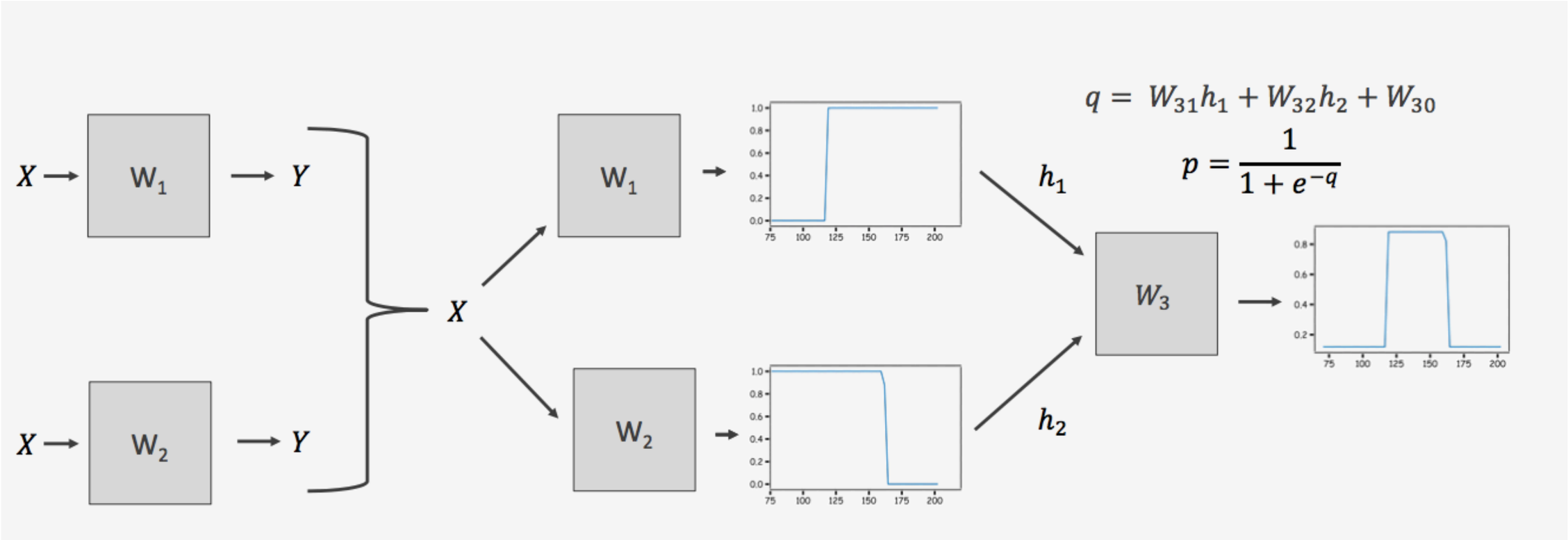
```
for i in range(nb_epochs):
  np.random.shuffle(data)
  for example in data:
    params_grad = evaluate_gradient(loss_function, example, params)
    params = params - learning_rate * params_grad
```

Mini-Batch: do some at a time


Iteration 0
Cost
3000
2500
2000
1500
1000
500
0
40
60
80
100
120
Slope
15
10
5
0
−5
−10
−15
−20
−25
Intercept

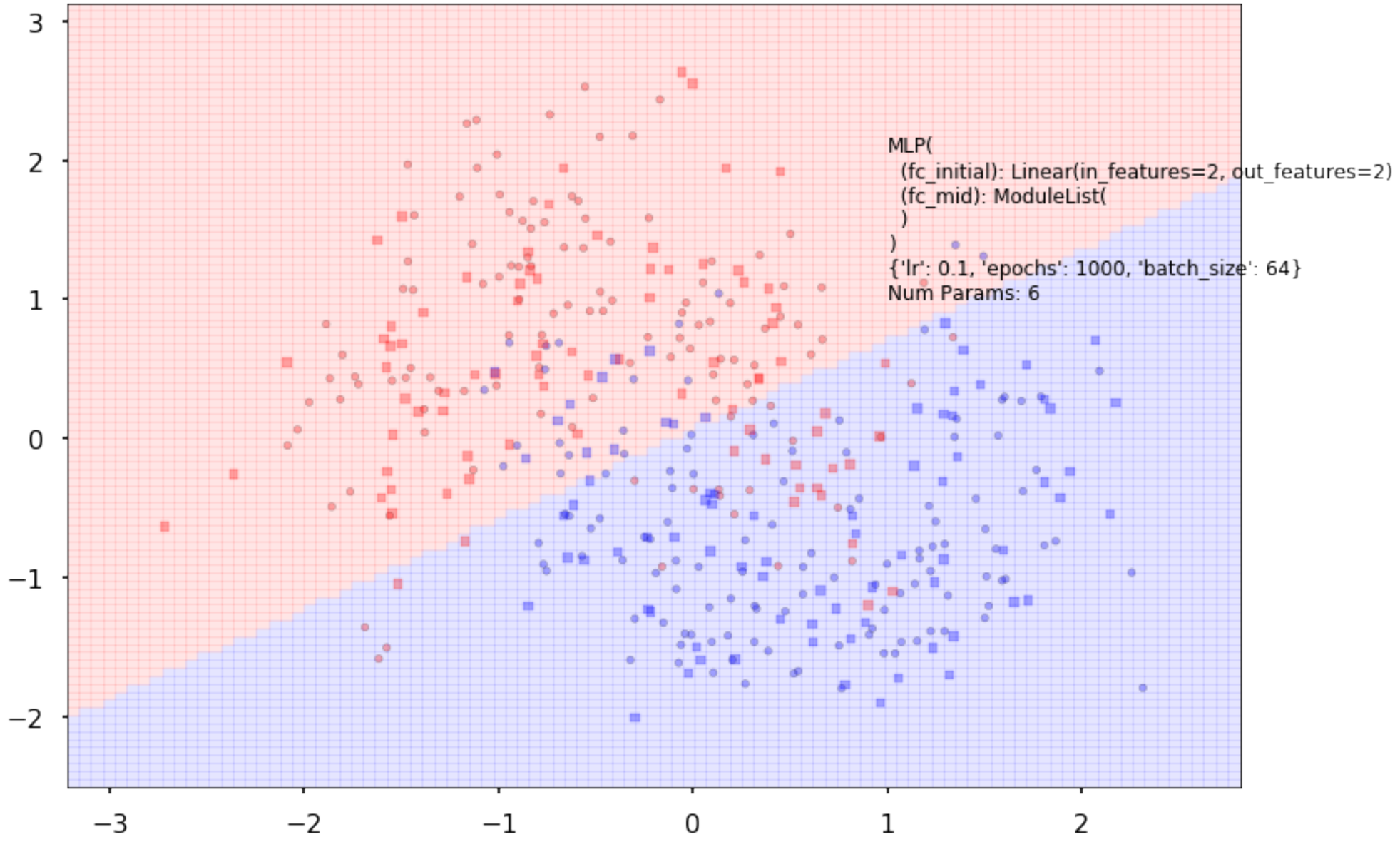

Full Likelihood:
Batch Likelihood:
L
θ

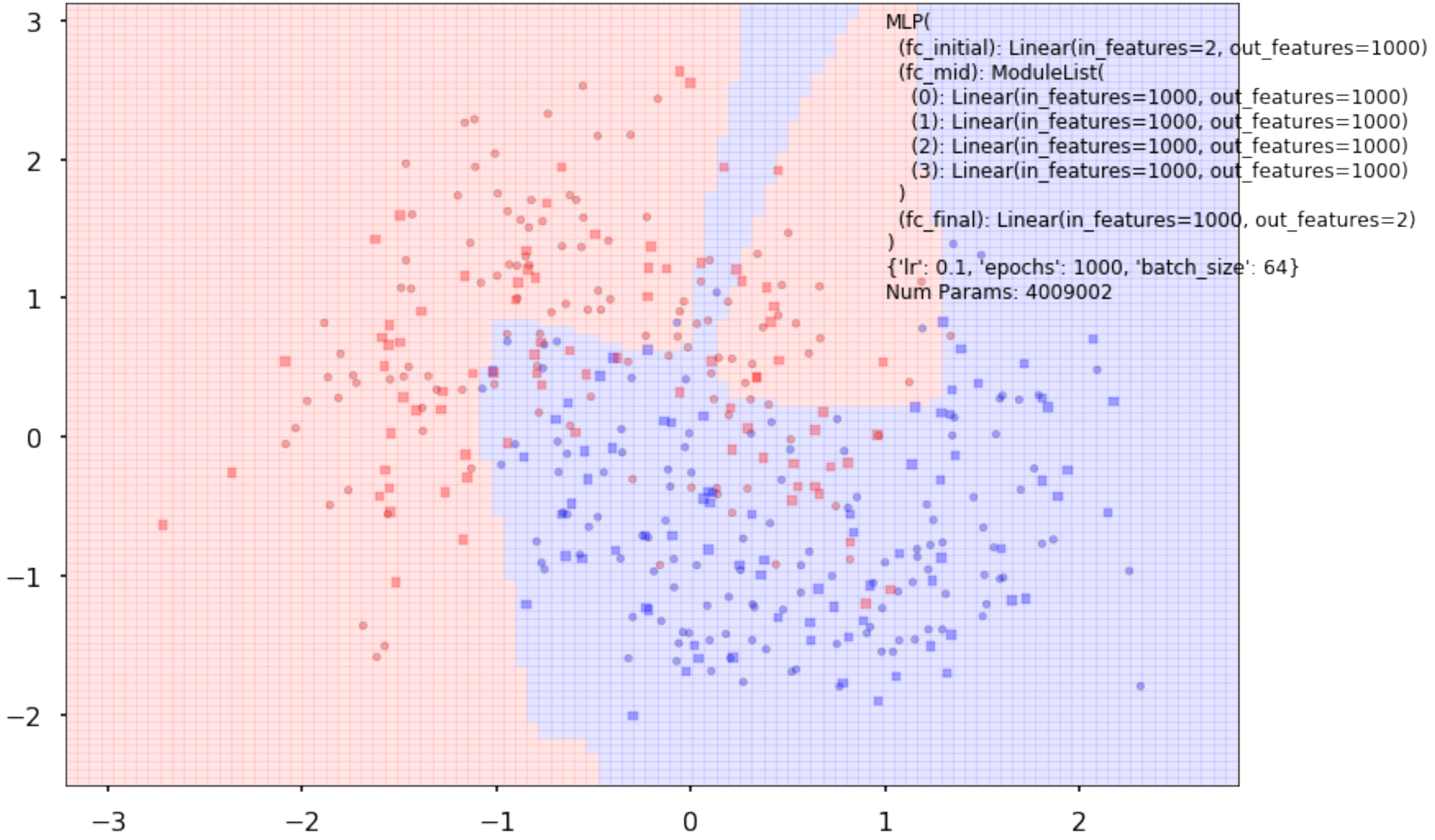
Basic Idea: Universal approx by combining nonlinears

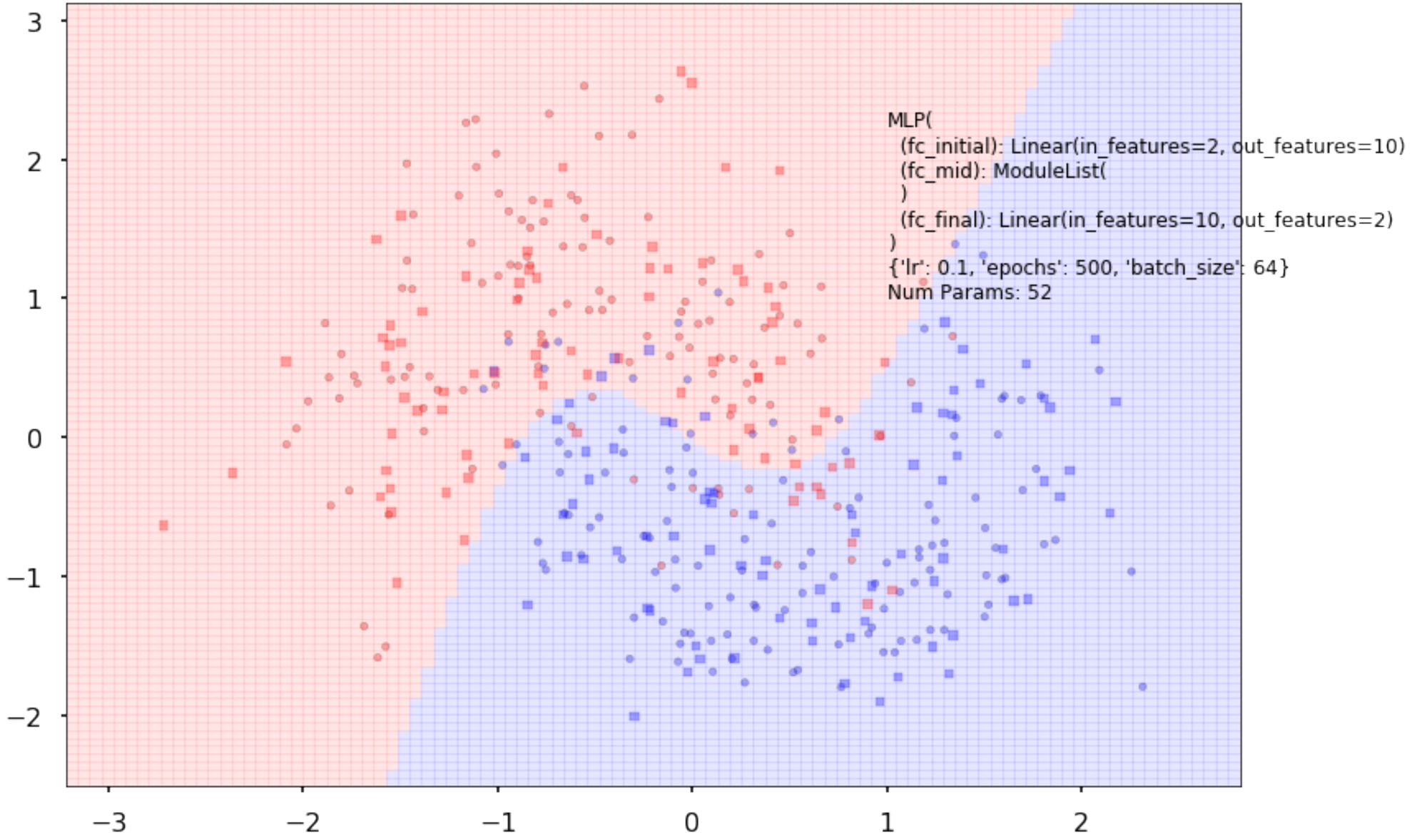


Multi Layer Perceptrons

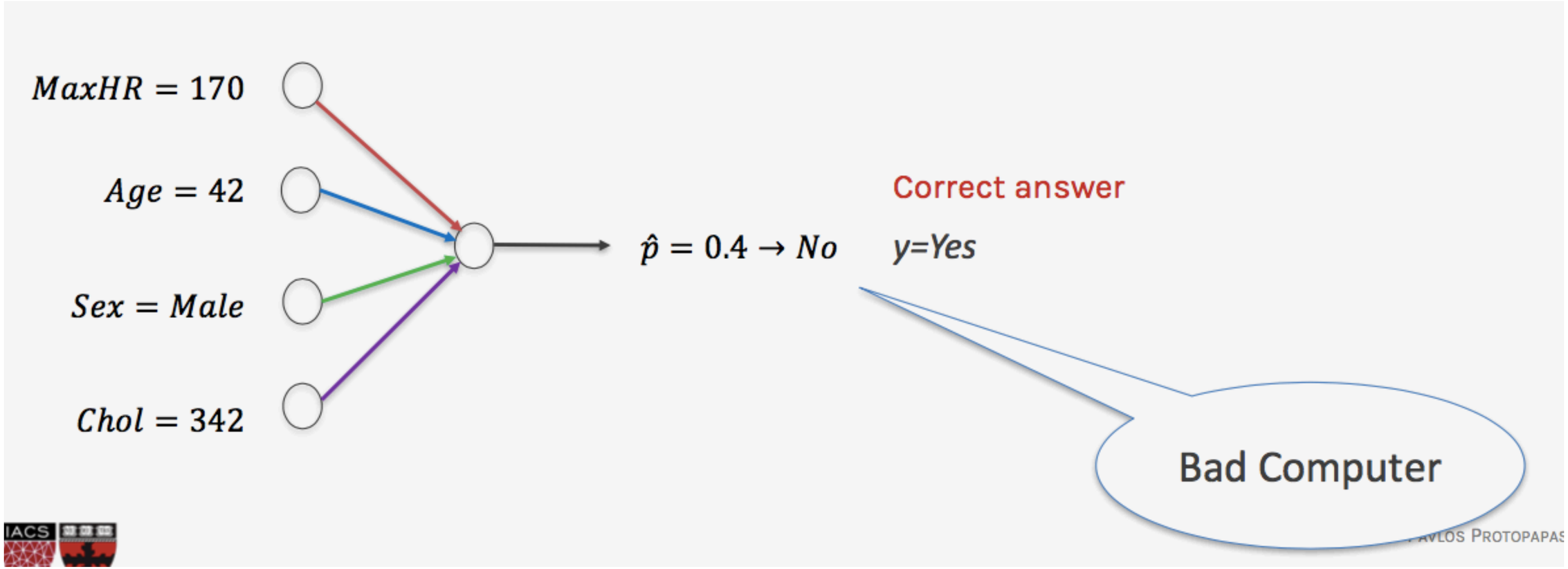
```
# create model
model = Sequential()
model.add(Flatten(input_shape=(img_width, img_height)))
model.add(Dense(config.hidden_nodes, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer=config.optimizer,
              metrics=['accuracy'])
model.summary()
# Fit the model
model.fit(X_train, y_train, validation_data=(X_test, y_test),
          epochs=config.epochs,
          callbacks=[WandbCallback(data_type="image", labels=labels)])
```


MLP(
(fc_initial): Linear(in_features=2, out_features=2)
(fc_mid): ModuleList(
)
)
{'lr': 0.1, 'epochs': 1000, 'batch_size': 64}
Num Params: 6
3
2
1
0
−1
−2
−3
−2
−1
0
1
2

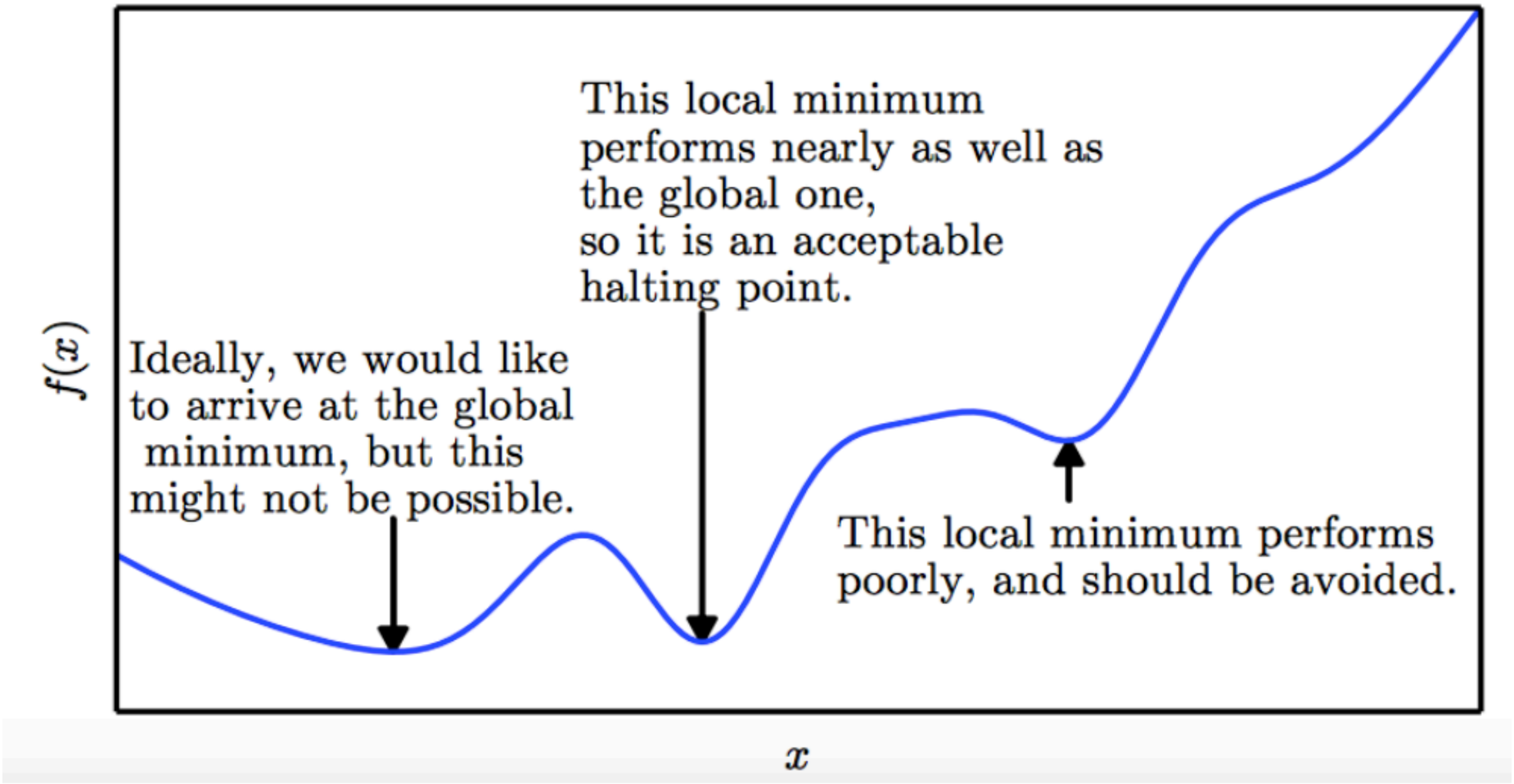

MLP(
(fc_initial): Linear(in_features=2, out_features=1000)
(fc_mid): ModuleList(
(0): Linear(in_features=1000, out_features=1000)
(1): Linear(in_features=1000, out_features=1000)
(2): Linear(in_features=1000, out_features=1000)
(3): Linear(in_features=1000, out_features=1000)
)
(fc_final): Linear(in_features=1000, out_features=2)
)
{'lr': 0.1, 'epochs': 1000, 'batch_size': 64}
Num Params: 4009002
3
2
1
0
−1
−2
−3
−2
−1
0
1
2


MLP(
(fc_initial): Linear(in_features=2, out_features=10)
(fc_mid): ModuleList(
)
(fc_final): Linear(in_features=10, out_features=2)
)
{'lr': 0.1, 'epochs': 500, 'batch_size': 64}
Num Params: 52
3
2
1
0
−1
−2
−3
−2
−1
0
1
2

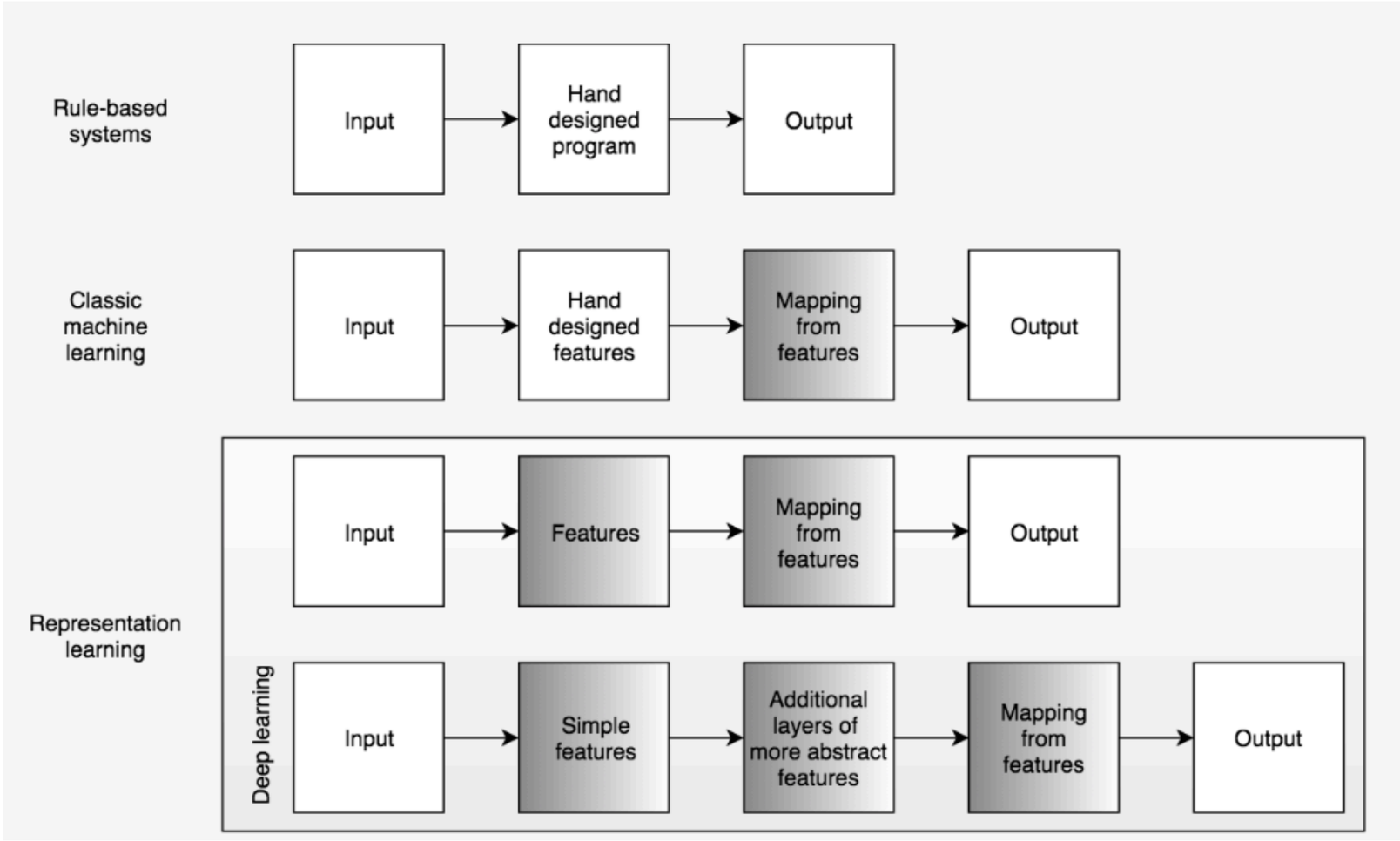
We start with Garbage



And get better, but we dont need to be best



BIG IDEA: learn hierarchical representations



And we need to help it

- MLPs are crude
- we must help computer
- thus use CNNs and RNNs which direct the representations
- directed representation learning is the basis for transfer learning

Three pillars

GPU

Automatic Differentiation

Representation Learning

Statistics: Likelihoods

Output Type	Output Distribution	Output layer	Cost Function
Binary	Bernoulli	Sigmoid	Binary Cross Entropy
Discrete	Multinoulli	Softmax	Cross Entropy
Continuous	Gaussian	Linear	MSE
Continuous	Arbitrary	-	GANS

Practicalities

Problem: Gradient vanishing and explosion

$$\text{Suppose } \mathbf{W} = \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}:$$

$$\begin{bmatrix} h_1^1 \\ h_2^1 \end{bmatrix} = \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad \cdots \quad \begin{bmatrix} h_1^n \\ h_2^n \end{bmatrix} = \begin{bmatrix} a^n & 0 \\ 0 & b^n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Mitigations and Optimization

- Explosion: Gradient clipping
- Implosion: Skip Connections, Resnets, LSTM, Attention
- Curvature: Momentum, Gradient Accumulation
- Heterogeneity: adaptive learning rates

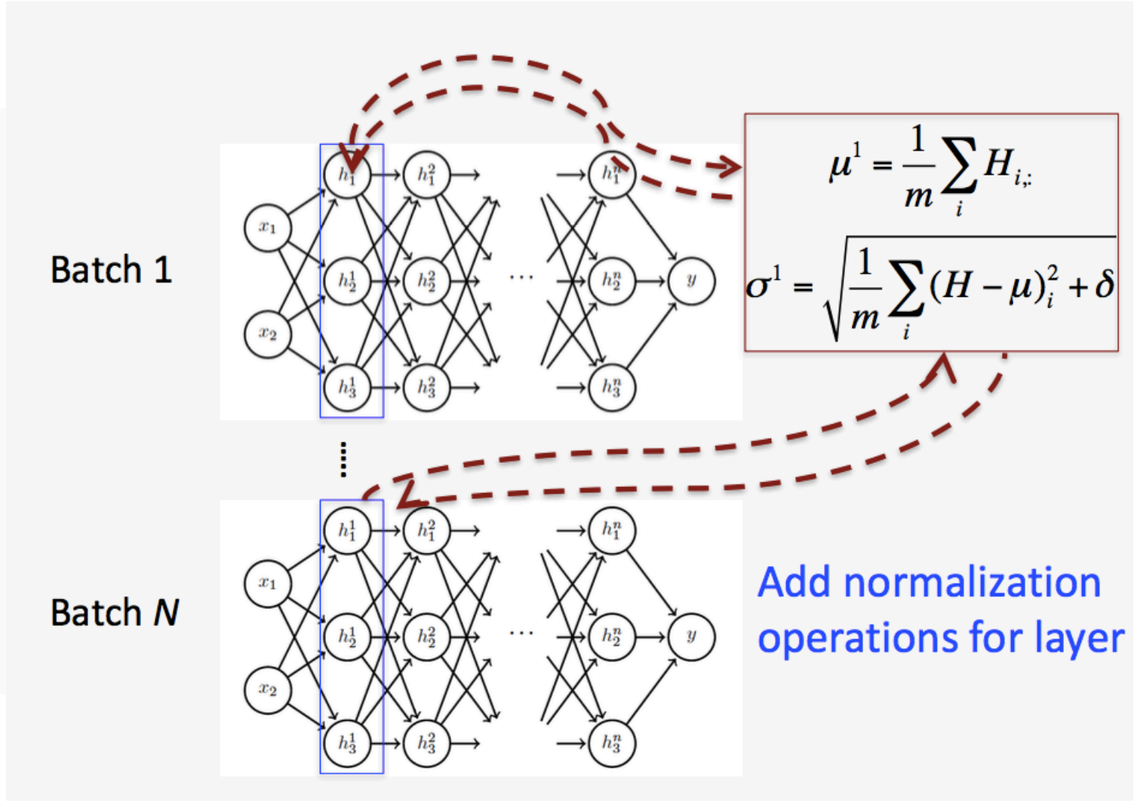
Initialization

- Uniform or Normal: get unit variance
- Non saturated Bias for Relus
- Feature Normalization
- Batch Norm

Batch Norm

$$\gamma H' + \beta$$

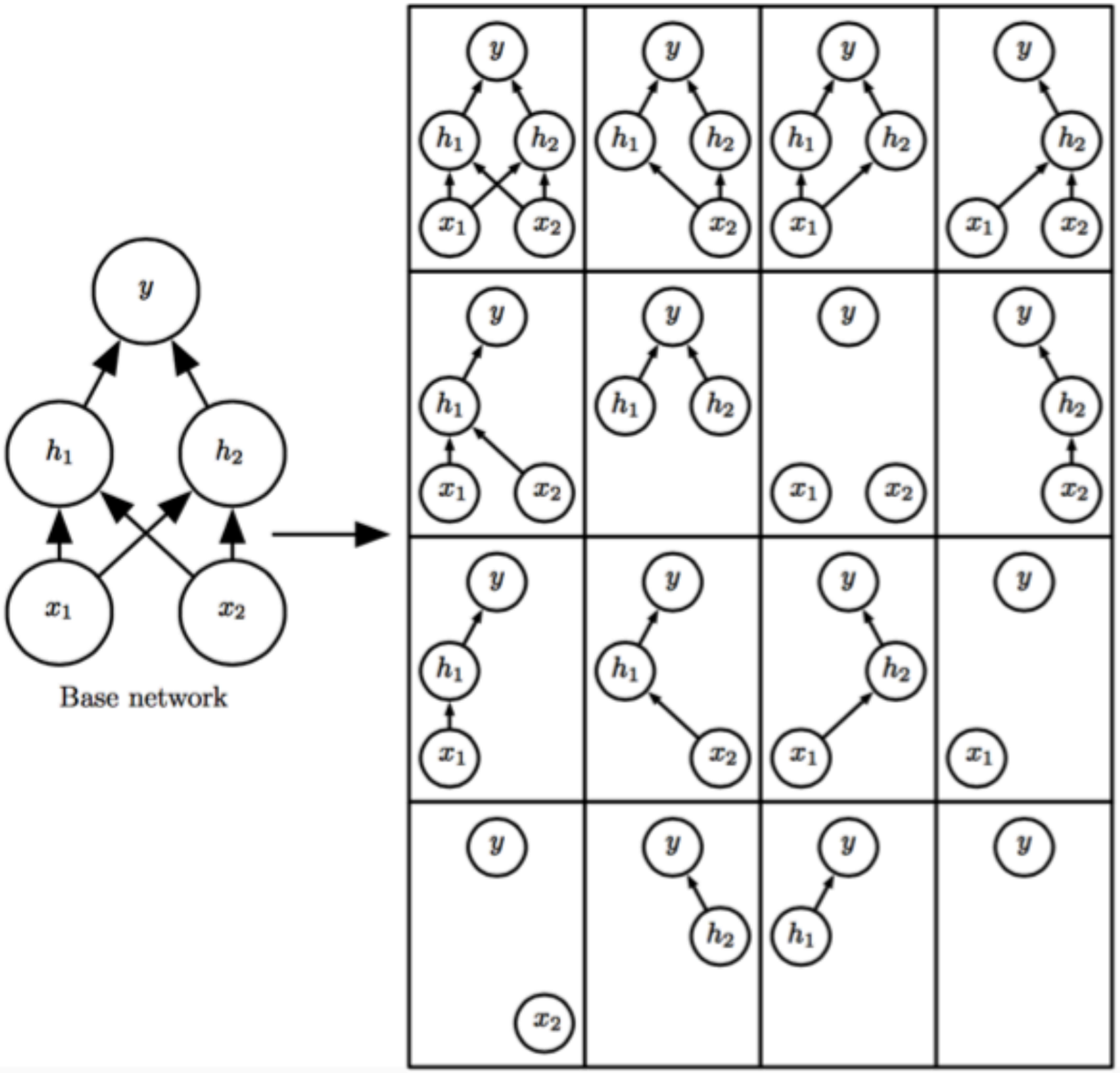
Learnable parameters



First Overfit

Dropout

```
model=Sequential()
model.add(Flatten(input_shape=(img_width,img_height)))
model.add(Dropout(config.dropout))
model.add(Dense(config.hidden_nodes, activation='relu'))
model.add(Dropout(config.dropout))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer=config.optimizer,
                    metrics=['accuracy'])
```



L1/L2: Weight Decay

We used to optimize:

$$W^{(i+1)} = W^{(i)} - \lambda \frac{\partial J}{\partial W}$$

Change to ...

$$J_R(W; X, y) = J(W; X, y) + \frac{1}{2}\alpha W^2$$

$$W^{(i+1)} = W^{(i)} - \lambda \frac{\partial J}{\partial W} - \lambda\alpha\, W$$

weights decay in proportion to its size.

Biases not penalized

L_2 regularization:

- **Decay of weights**
- MAP estimation with Gaussian prior

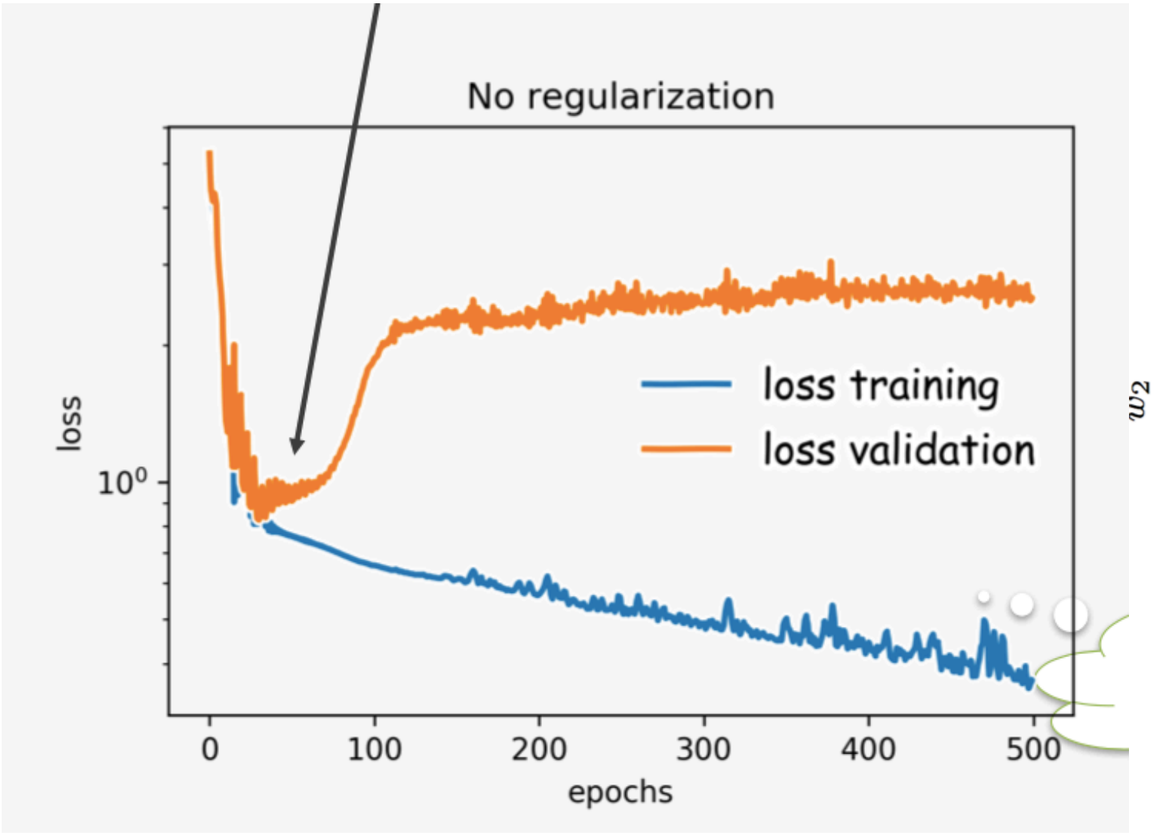
$$\Omega(W) = \frac{1}{2} \| W \|_2^2$$

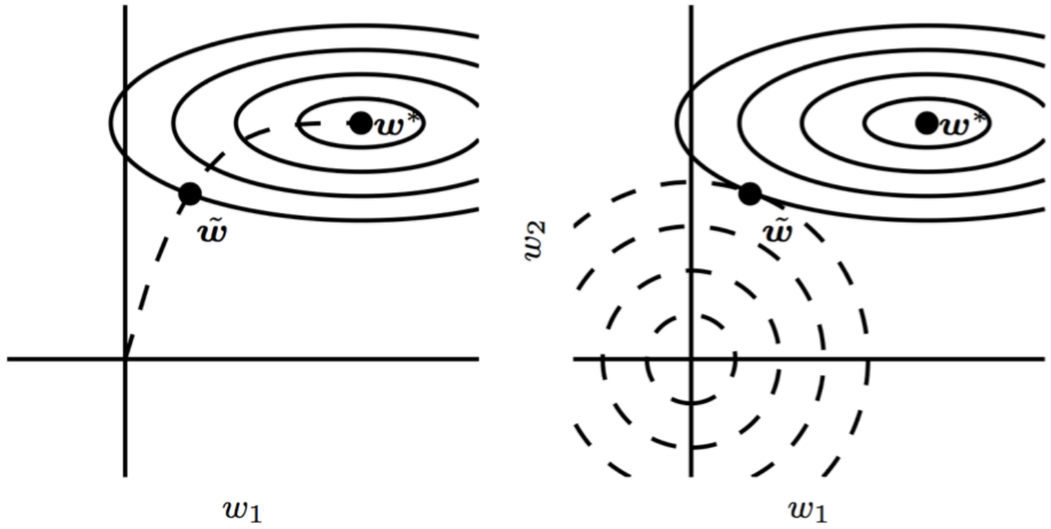
L_1 regularization:

- encourages sparsity
- MAP estimation with Laplacian prior

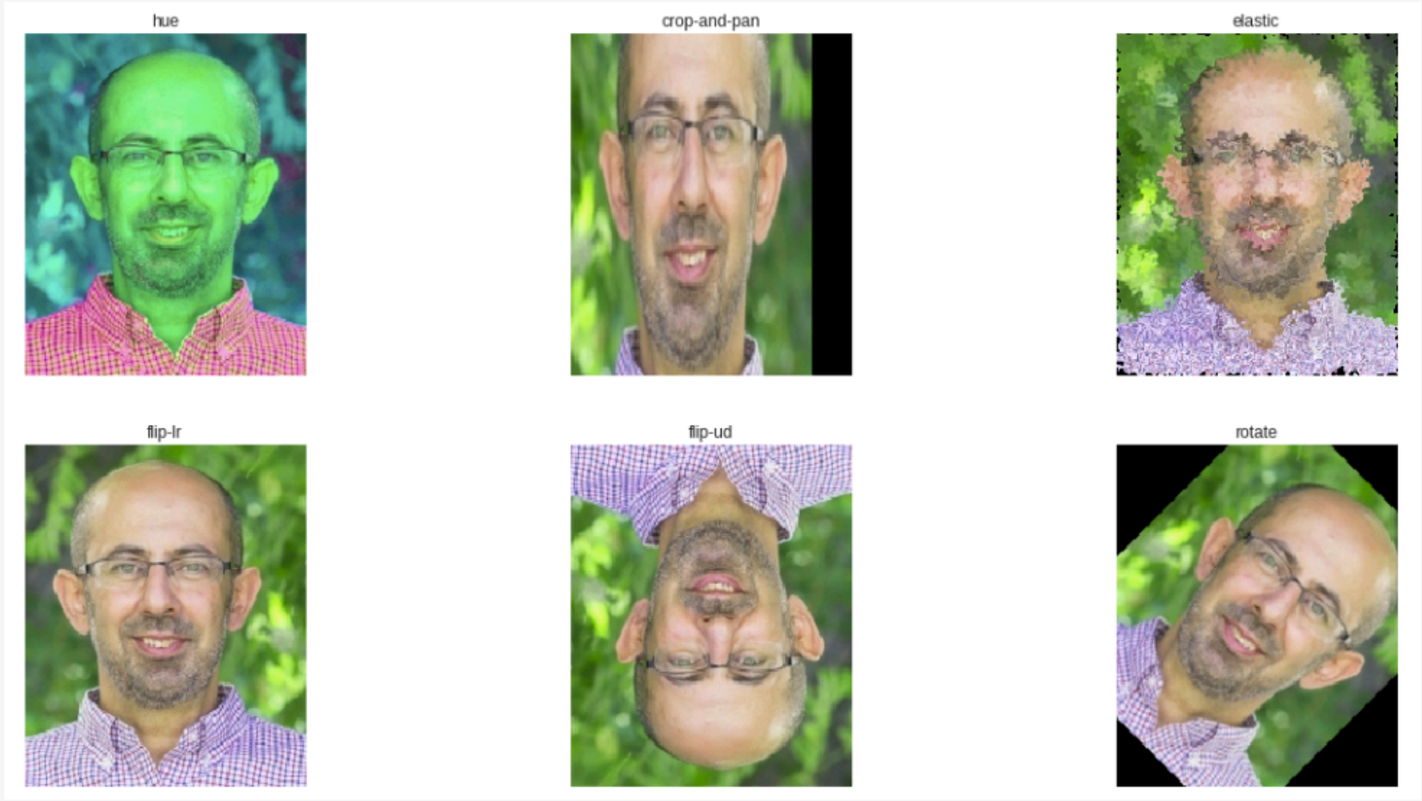
$$\Omega(W) = \frac{1}{2} \| W \|_1$$

Early Stopping





Data Aug

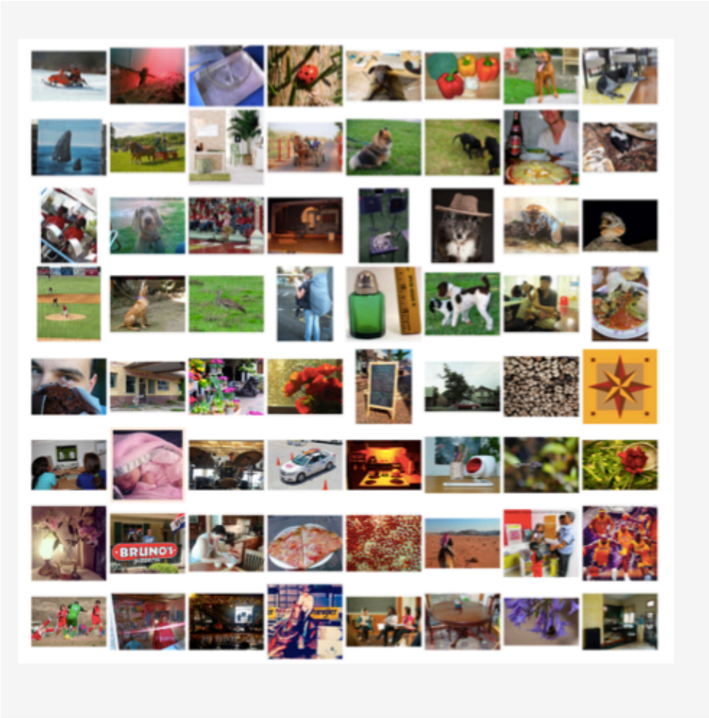


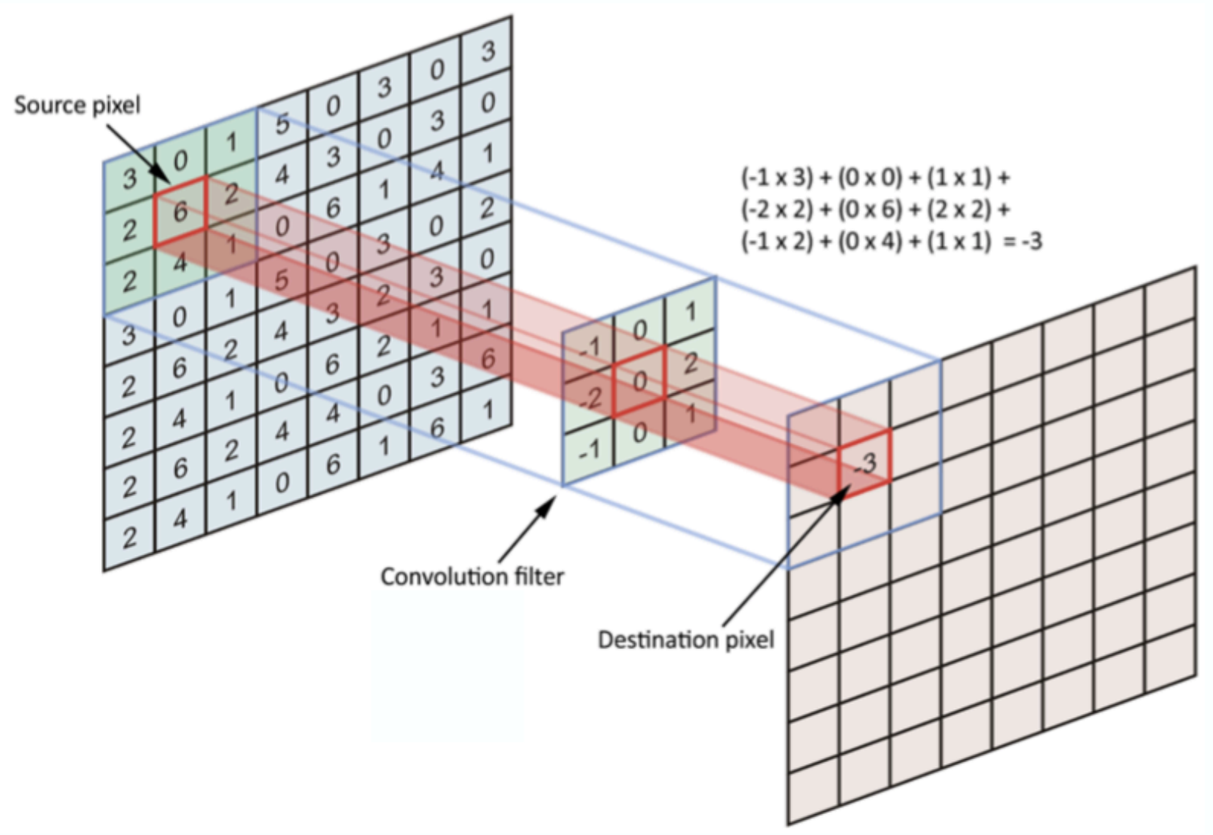
Others

- Bagging
- Adverserial Examples
- Sparse Activations (on activations not weights)

Convolutional Neural Networks

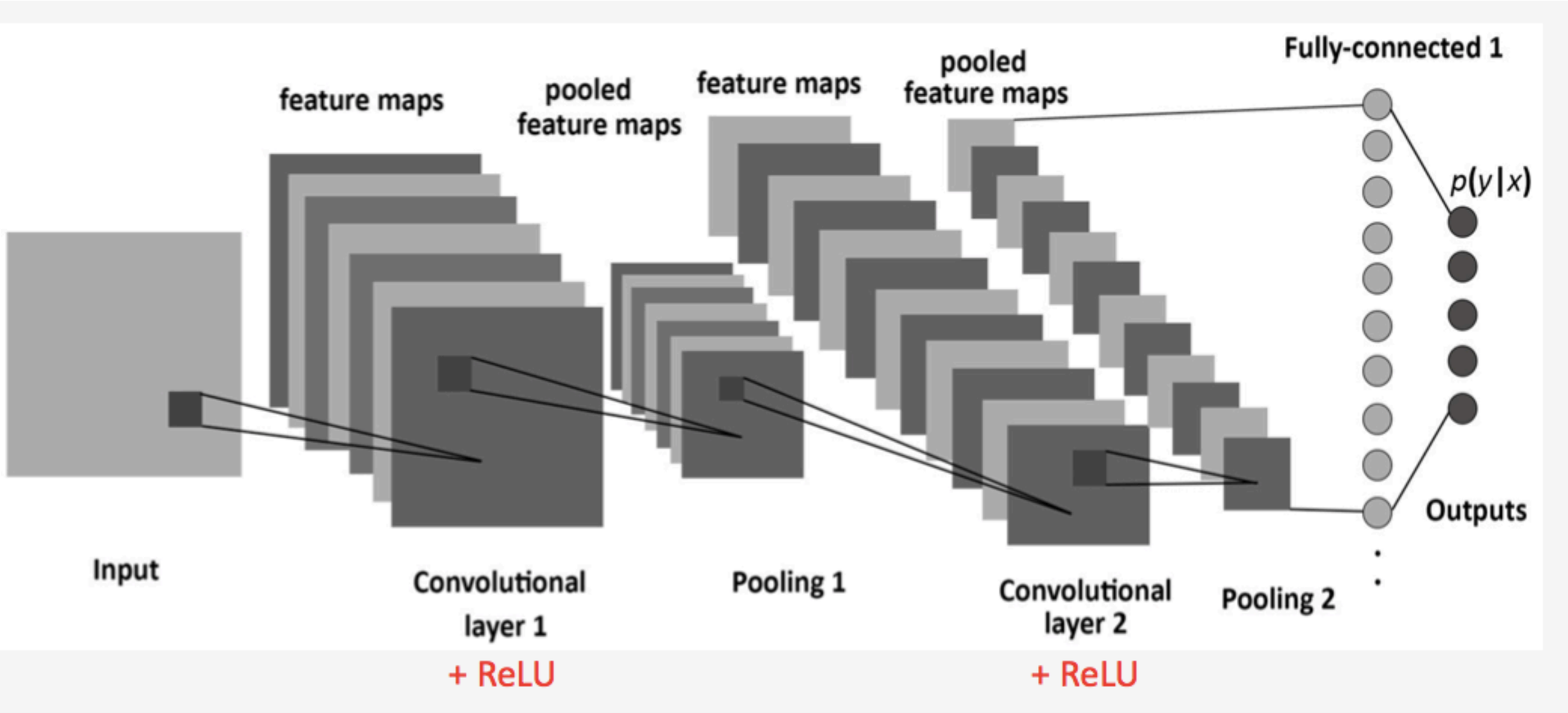
Exploding params: do weight tying



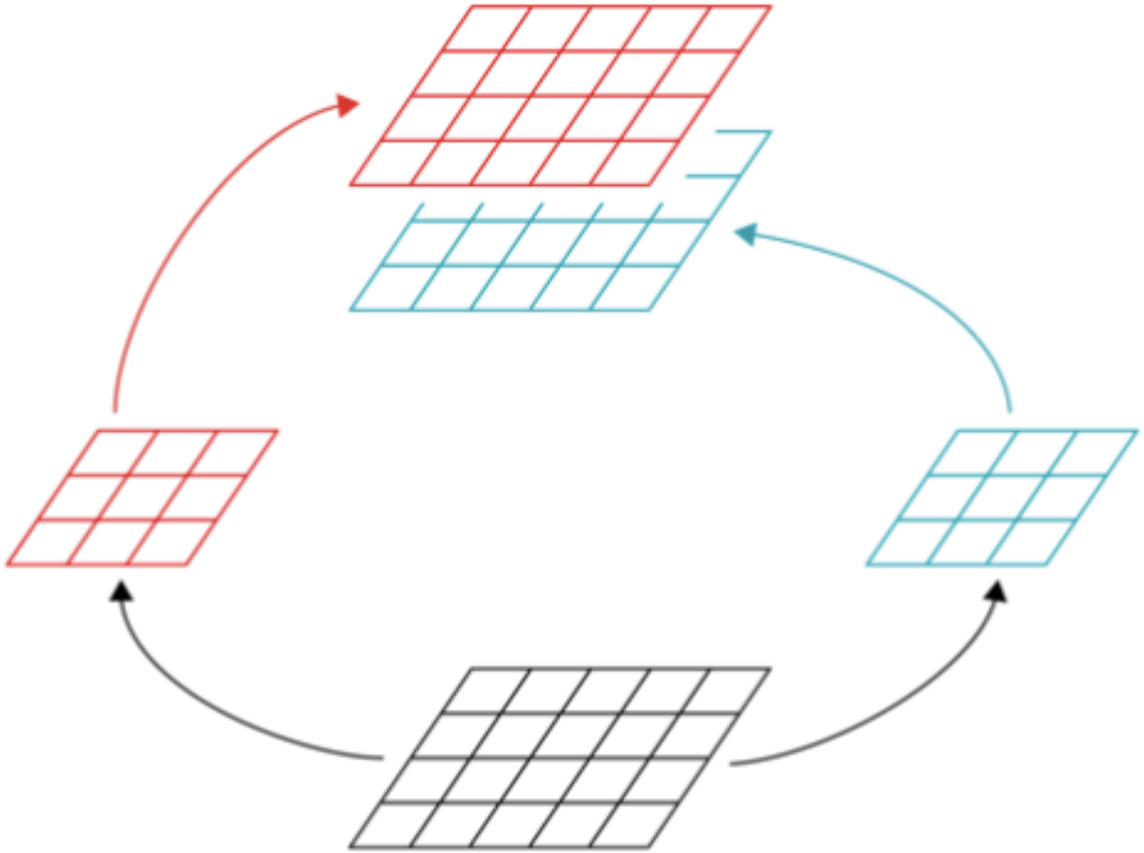


- 3073 params cifar-10
- 150129 Imagenet
- Nearer pizels are related
- And there are symmetries

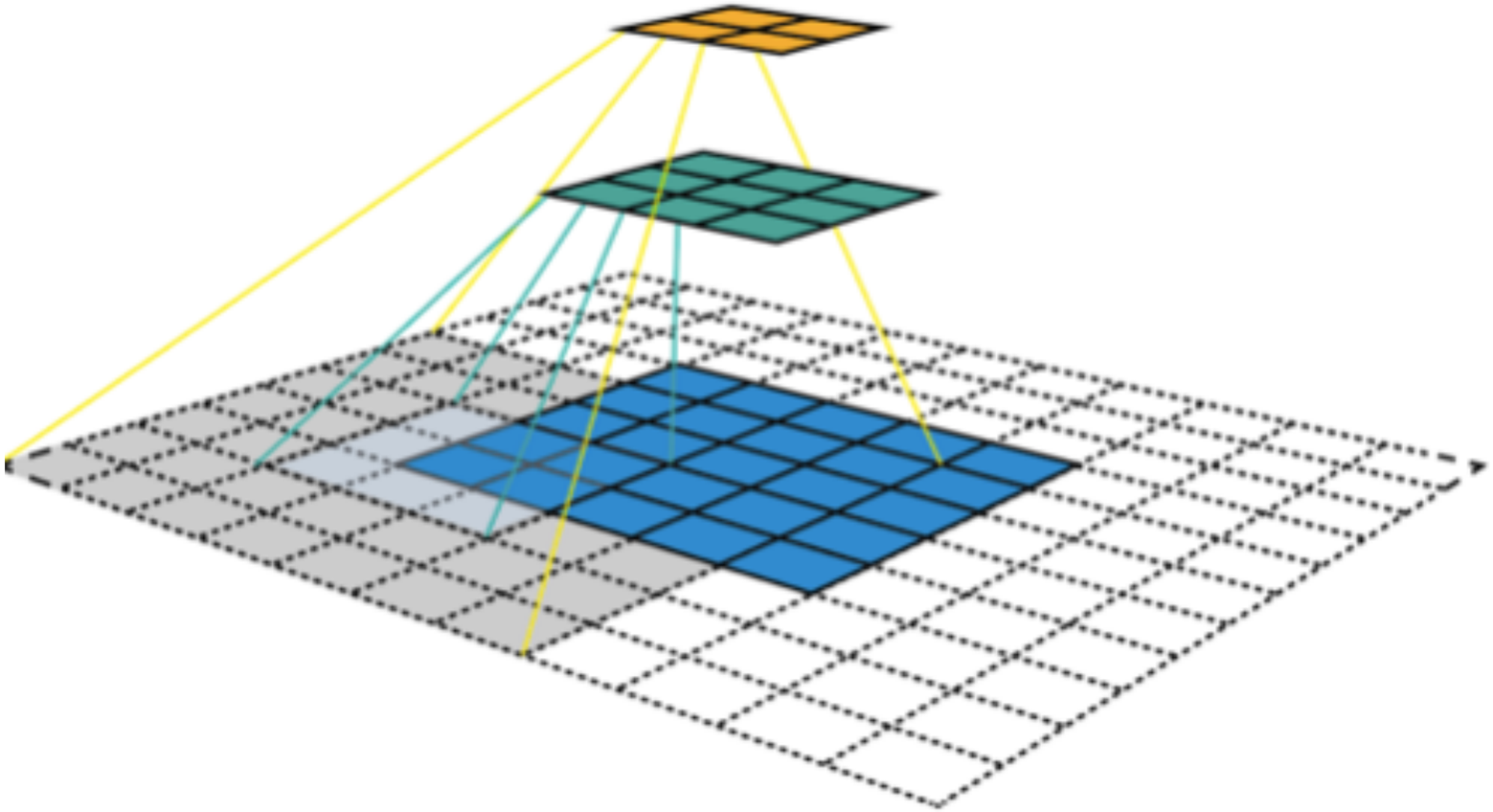
Architecture



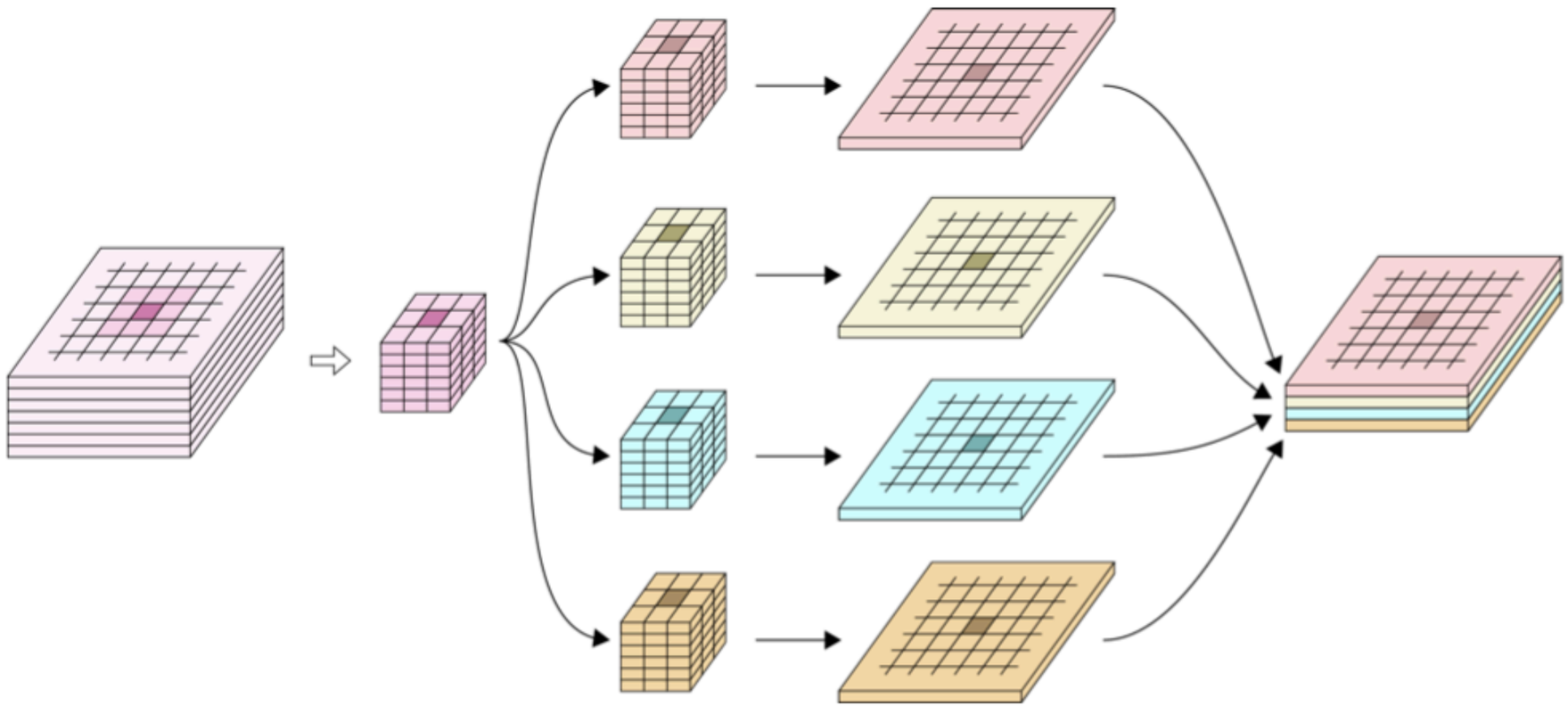
Featuremaps



Receptive Field



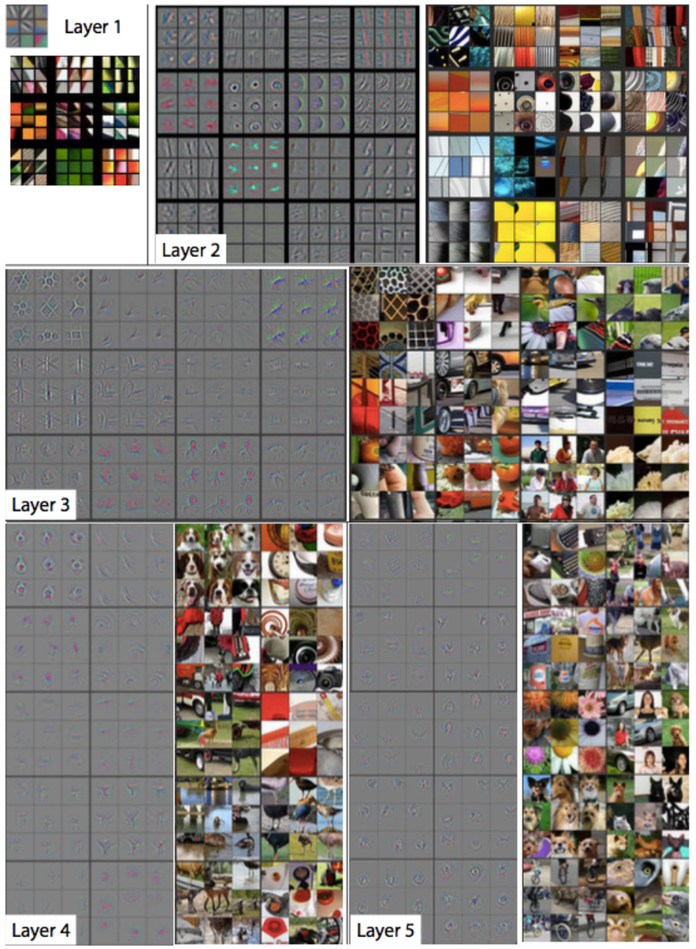
How fetauremaps in filters work



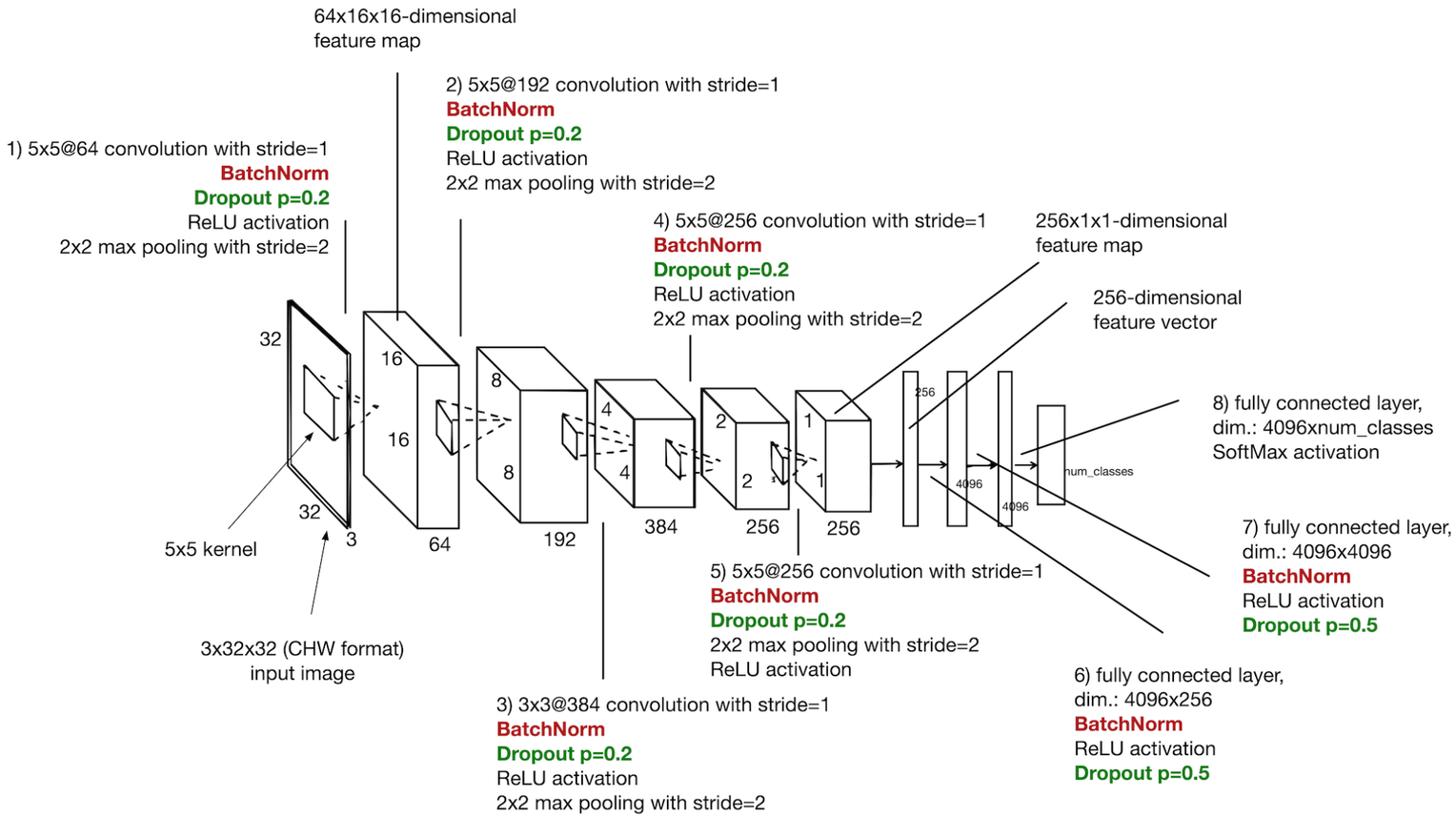
Things to do

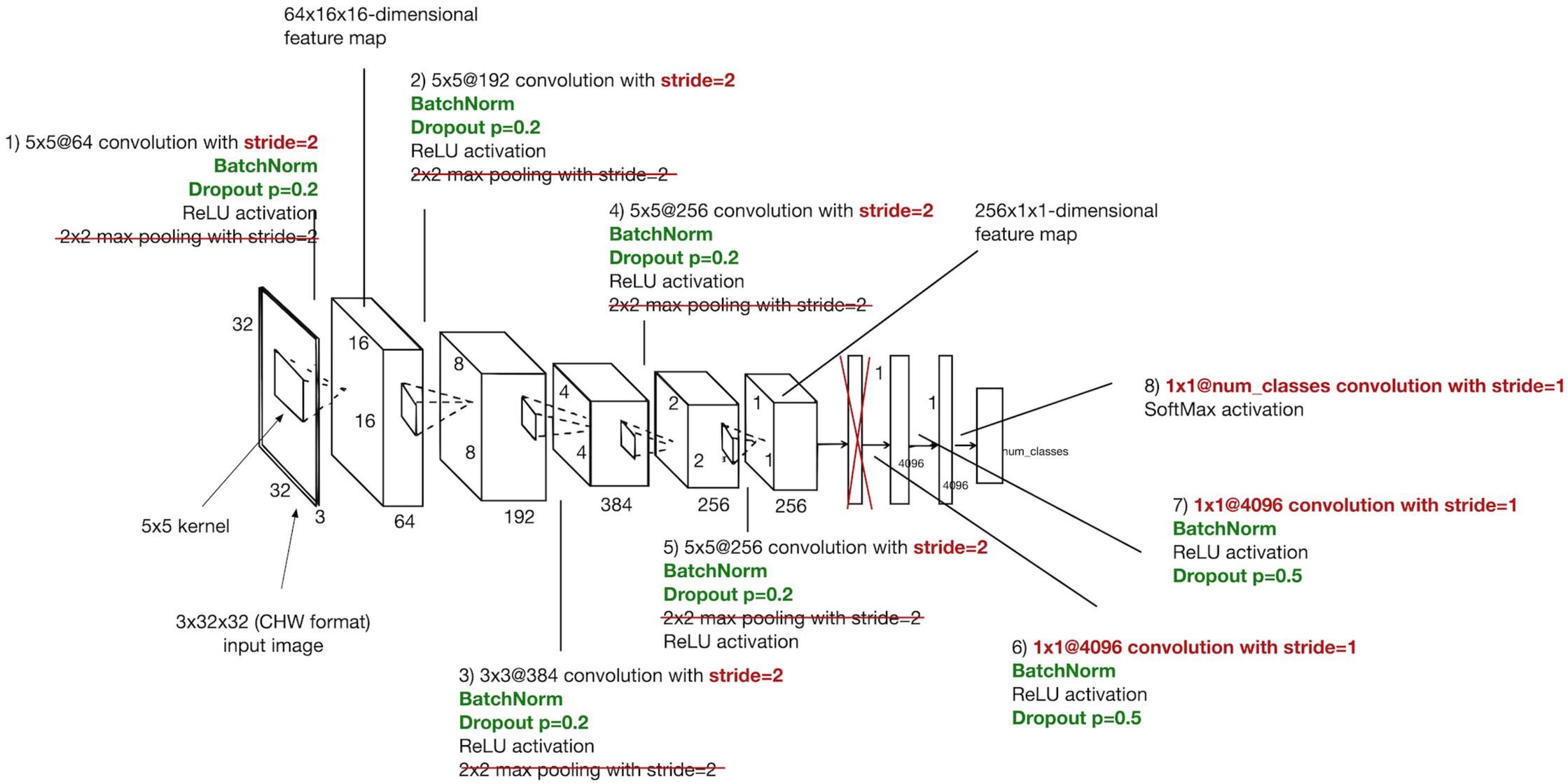
- striding
- padding
- pooling
- upsampling
- 1x1 convolutions
- globalavgpooling
- reshaping, Densing

Recursive Learning

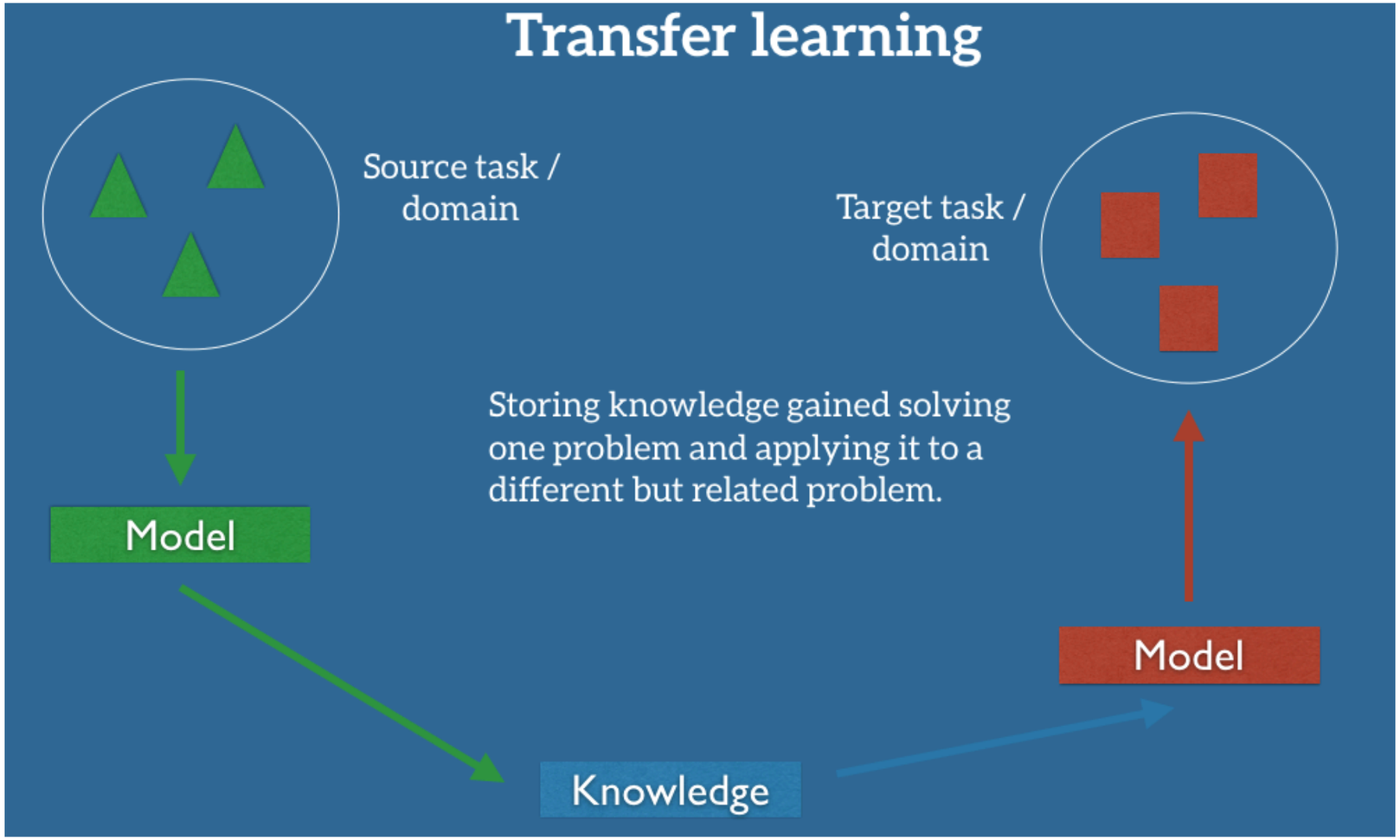


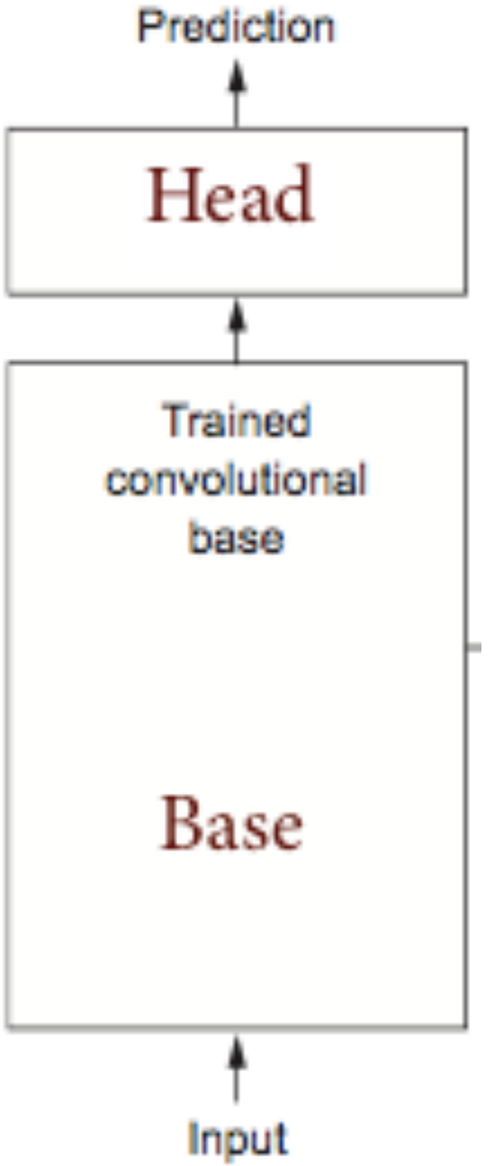
Architectures

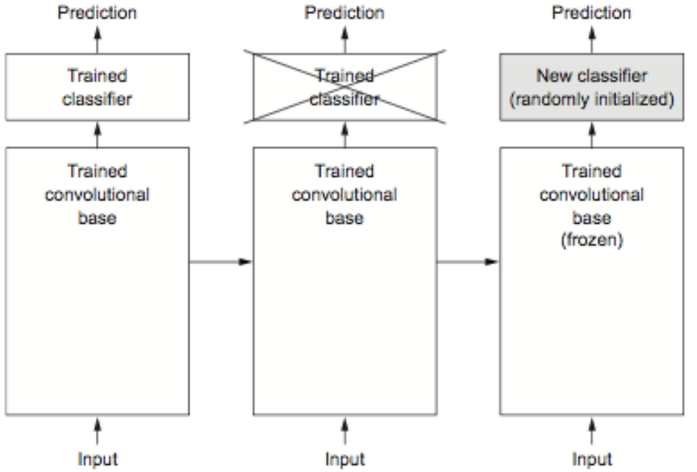


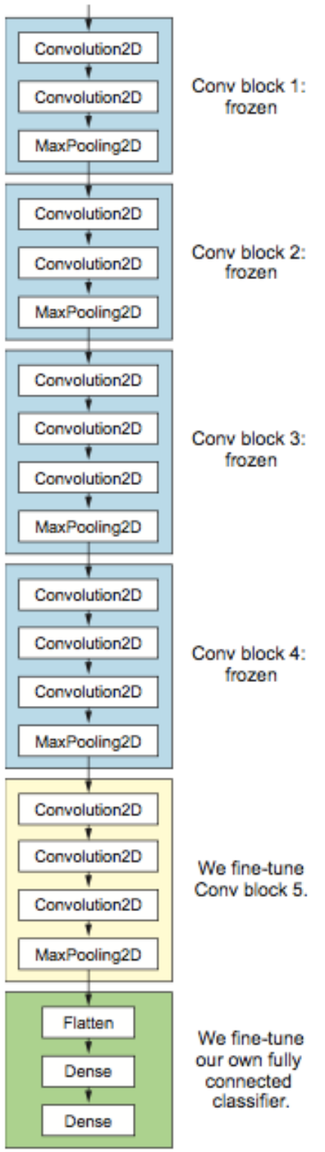

64x16x16-dimensional
feature map
2) 5x5@192 convolution with stride=2
BatchNorm
Dropout p=0.2
ReLU activation
2x2 max pooling with stride=2
1) 5x5@64 convolution with stride=2
BatchNorm
Dropout p=0.2
ReLU activation
2x2 max pooling with stride=2
4) 5x5@256 convolution with stride=2
BatchNorm
Dropout p=0.2
ReLU activation
2x2 max pooling with stride=2
256x1x1-dimensional
feature map
8) 1x1@num_classes convolution with stride=1
SoftMax activation
7) 1x1@4096 convolution with stride=1
BatchNorm
ReLU activation
Dropout p=0.5
6) 1x1@4096 convolution with stride=1
BatchNorm
ReLU activation
Dropout p=0.5
5) 5x5@256 convolution with stride=2
BatchNorm
Dropout p=0.2
2x2 max pooling with stride=2
ReLU activation
3) 3x3@384 convolution with stride=2
BatchNorm
Dropout p=0.2
ReLU activation
2x2 max pooling with stride=2
5x5 kernel
3x32x32 (CHW format)
input image
32
32
3
16
16
64
8
8
192
4
4
384
2
2
256
1
1
256
1
4096
1
4096
num_classes

Transfer Learning




Prediction
Head
Trained convolutional base
Base
Input


Prediction
Trained classifier
Trained convolutional base
Input
Prediction
Trained classifier
Trained convolutional base
Input
Prediction
New classifier (randomly initialized)
Trained convolutional base (frozen)
Input


Convolution2D
Convolution2D
MaxPooling2D
Conv block 1: frozen
Convolution2D
Convolution2D
MaxPooling2D
Conv block 2: frozen
Convolution2D
Convolution2D
Convolution2D
MaxPooling2D
Conv block 3: frozen
Convolution2D
Convolution2D
Convolution2D
MaxPooling2D
Conv block 4: frozen
Convolution2D
Convolution2D
Convolution2D
MaxPooling2D
We fine-tune Conv block 5.
Flatten
Dense
Dense
We fine-tune our own fully connected classifier.

Language Modeling + Embeddings

Basic

```
tokenizer = text.Tokenizer(num_words=config.vocab_size)
tokenizer.fit_on_texts(X_train)
X_train = tokenizer.texts_to_matrix(X_train)
X_test = tokenizer.texts_to_matrix(X_test)

bow_model = LogisticRegression()
bow_model.fit(X_train, y_train)

pred_train = bow_model.predict(X_train)
acc = np.sum(pred_train==y_train)/len(pred_train)

pred_test = bow_model.predict(X_test)
val_acc = np.sum(pred_test==y_test)/len(pred_test)
```

Language Modeling

Thou shalt not make a machine in the likeness of a human mind

Sliding window across running text

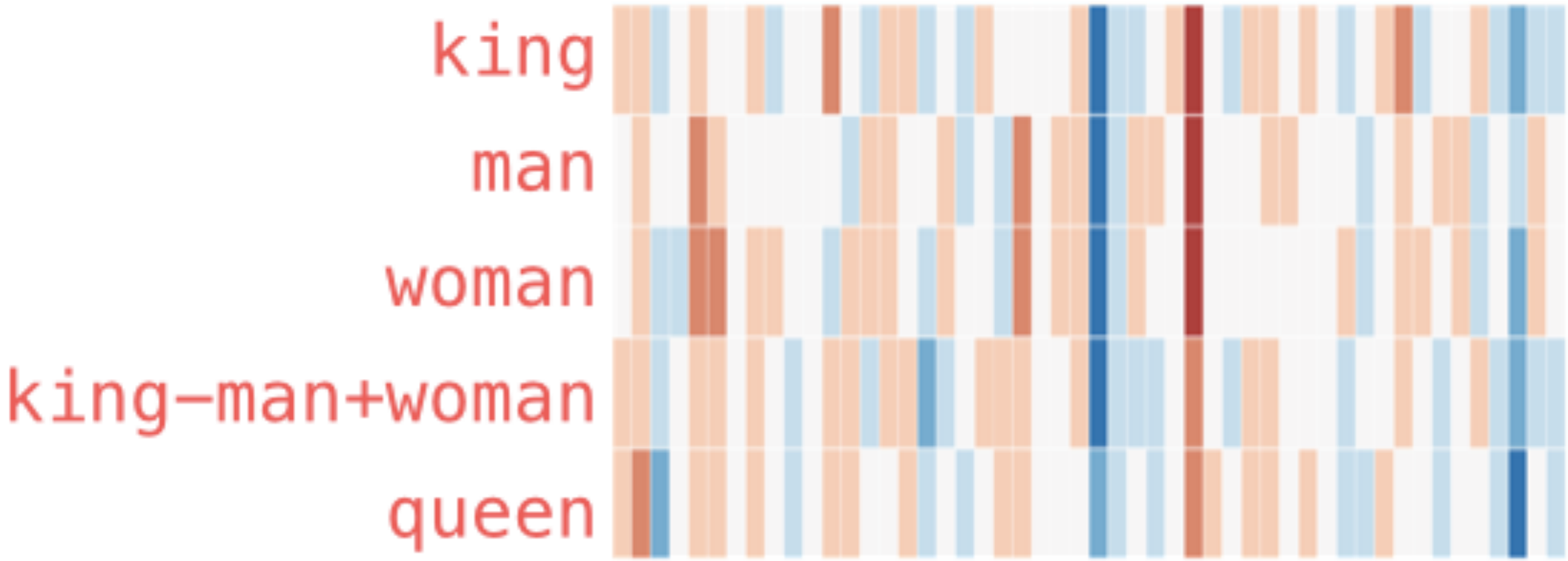
thou	shalt	not	make	a	machine	in	the	…
thou	shalt	not	make	a	machine	in	the	
thou	shalt	not	make	a	machine	in	the	
thou	shalt	not	make	a	machine	in	the	
thou	shalt	not	make	a	machine	in	the	

Dataset

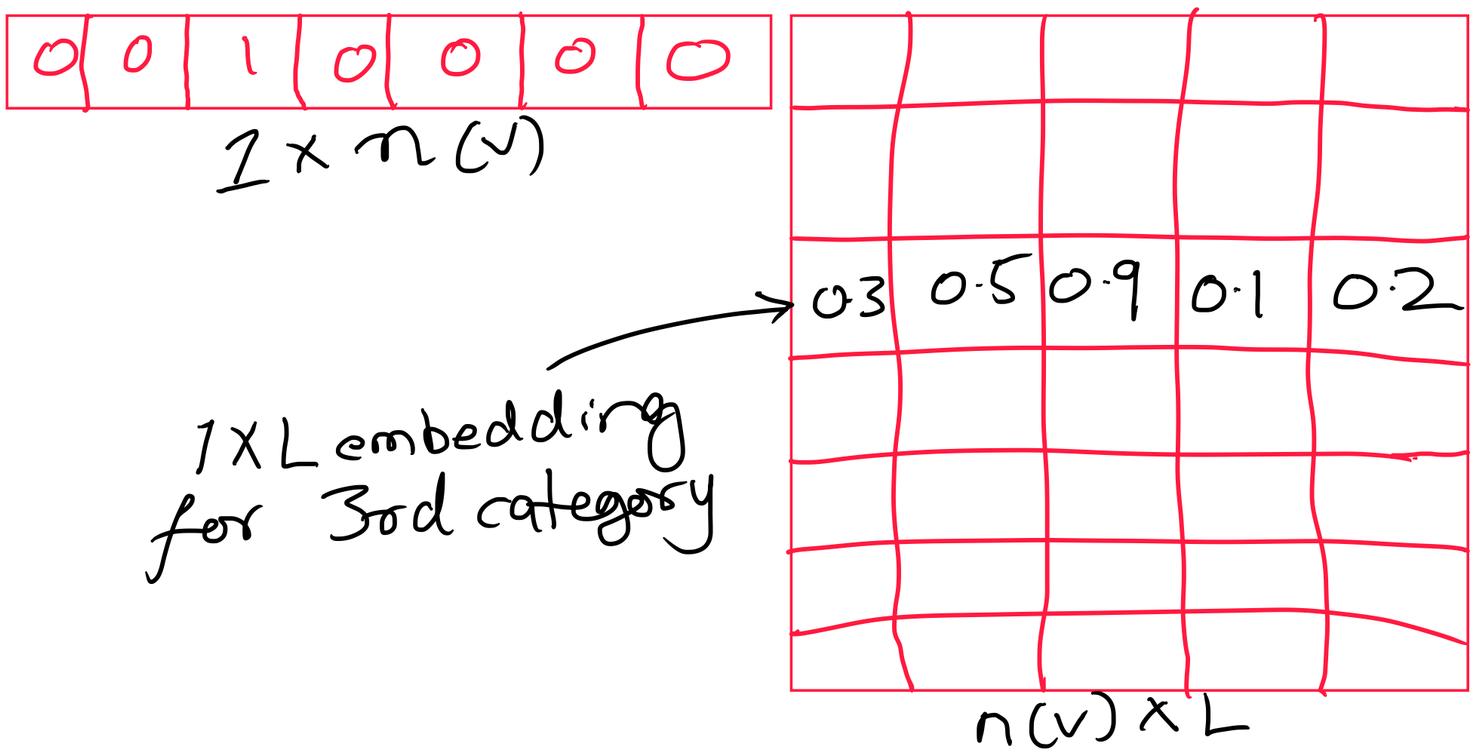
input 1	input 2	output
thou	shalt	not
shalt	not	make
not	make	a
make	a	machine
a	machine	in

Word Embeddings

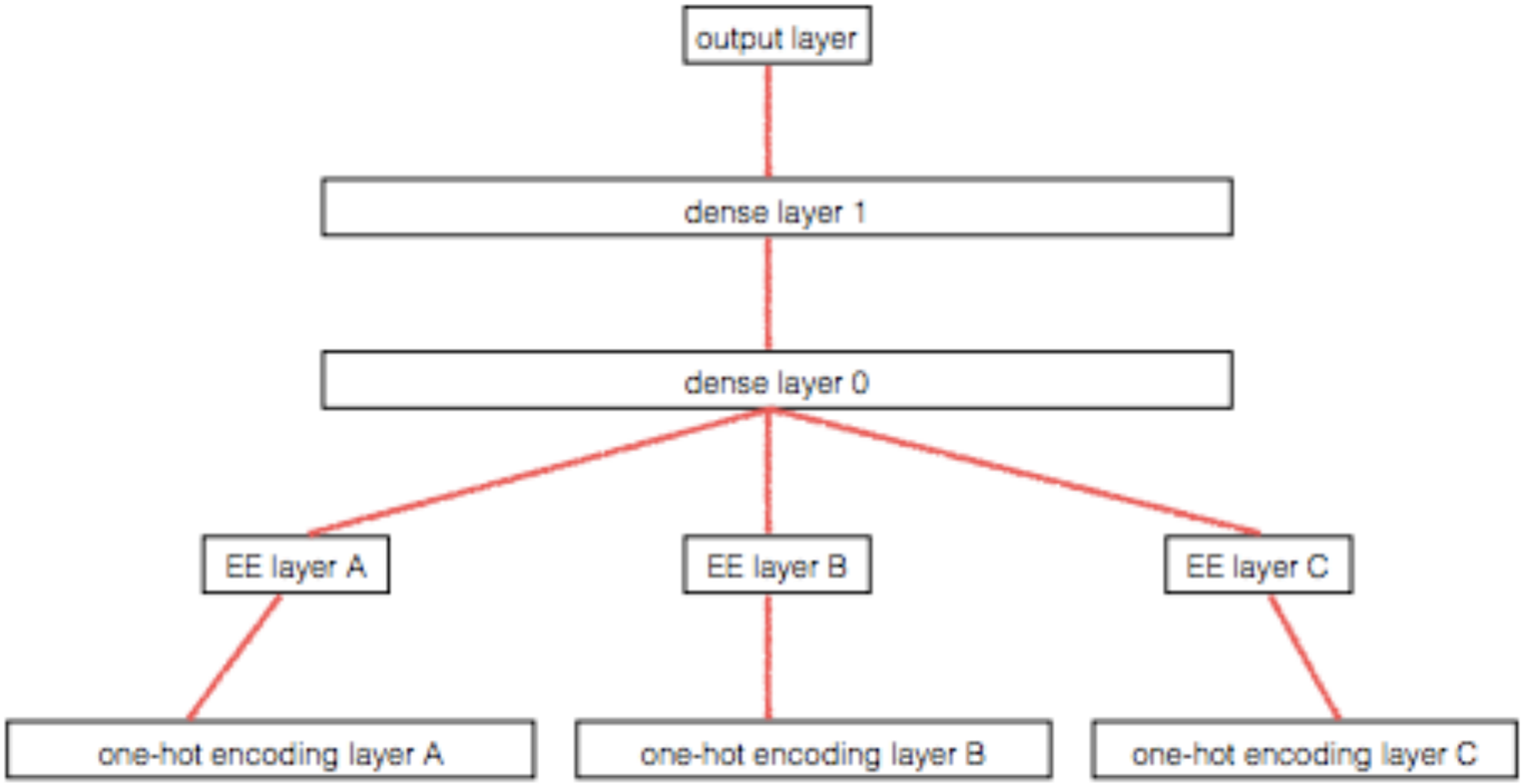
king − man + woman ~= queen



Embeddings are Linear Regression

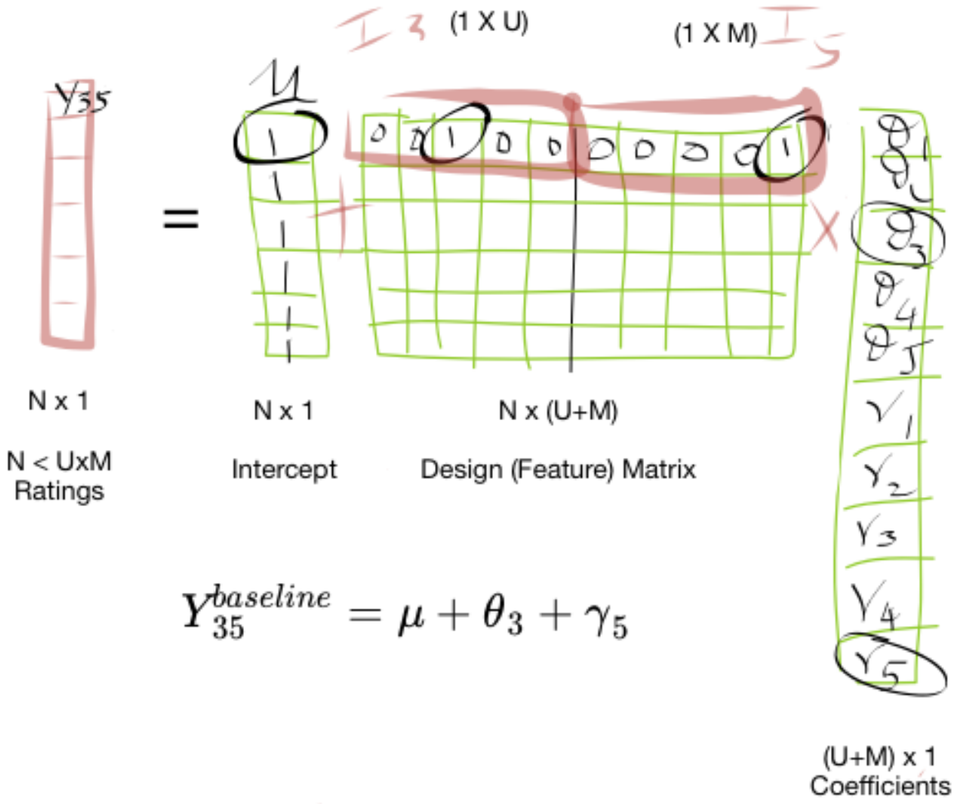


Learn Embeddings along with Task at hand



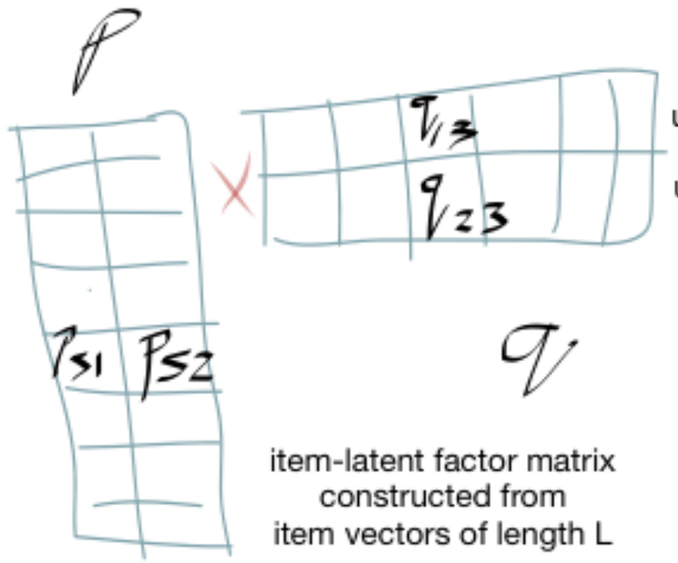
Application: Recommendations

$$Y_{35}^{baseline} = \mu + \bar{\theta} \cdot I_3 + \bar{\gamma} \cdot I_5 = \mu + [I_3, I_5] \cdot [\bar{\theta}, \bar{\gamma}]$$



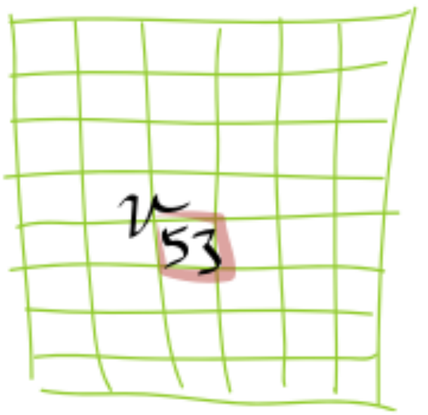
$$Y_{35}^{baseline} = \mu + \theta_3 + \gamma_5$$

L=2 latent factors



user-latent factor matrix
constructed from
user vectors of length L

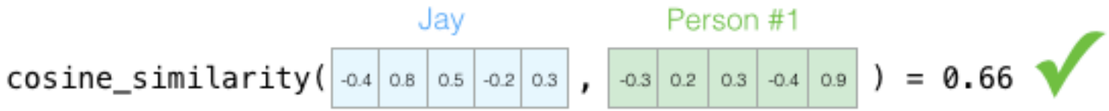
item-latent factor matrix
constructed from
item vectors of length L

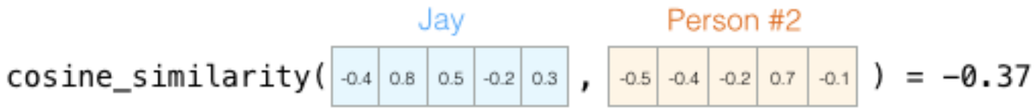


The product gives
the user-item
residual ranking matrix

The reasons for recommendations: similarity and FP

	Trait #1	Trait #2	Trait #3	Trait #4	Trait #5
Jay	-0.4	0.8	0.5	-0.2	0.3
Person #1	-0.3	0.2	0.3	-0.4	0.9
Person #2	-0.5	-0.4	-0.2	0.7	-0.1

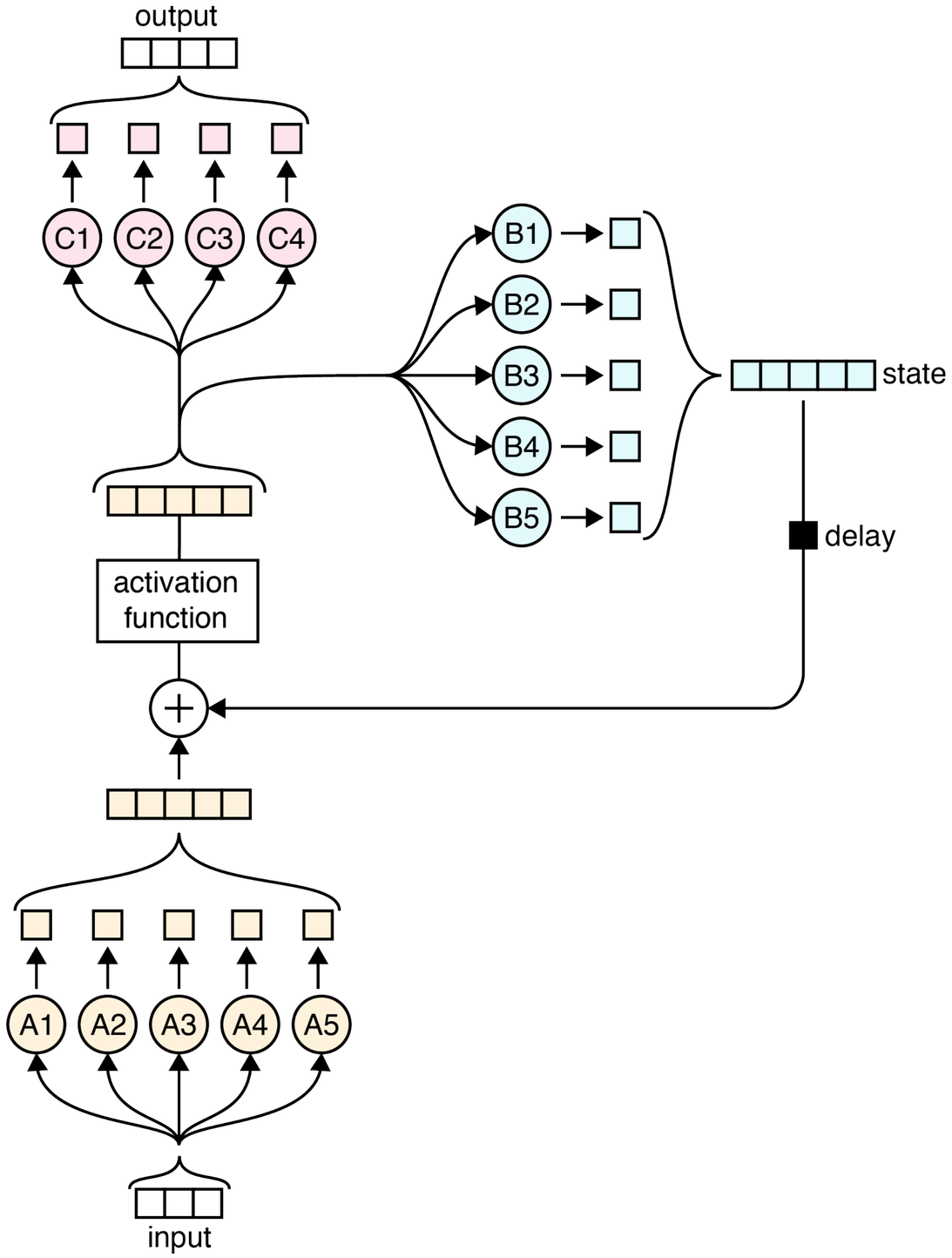




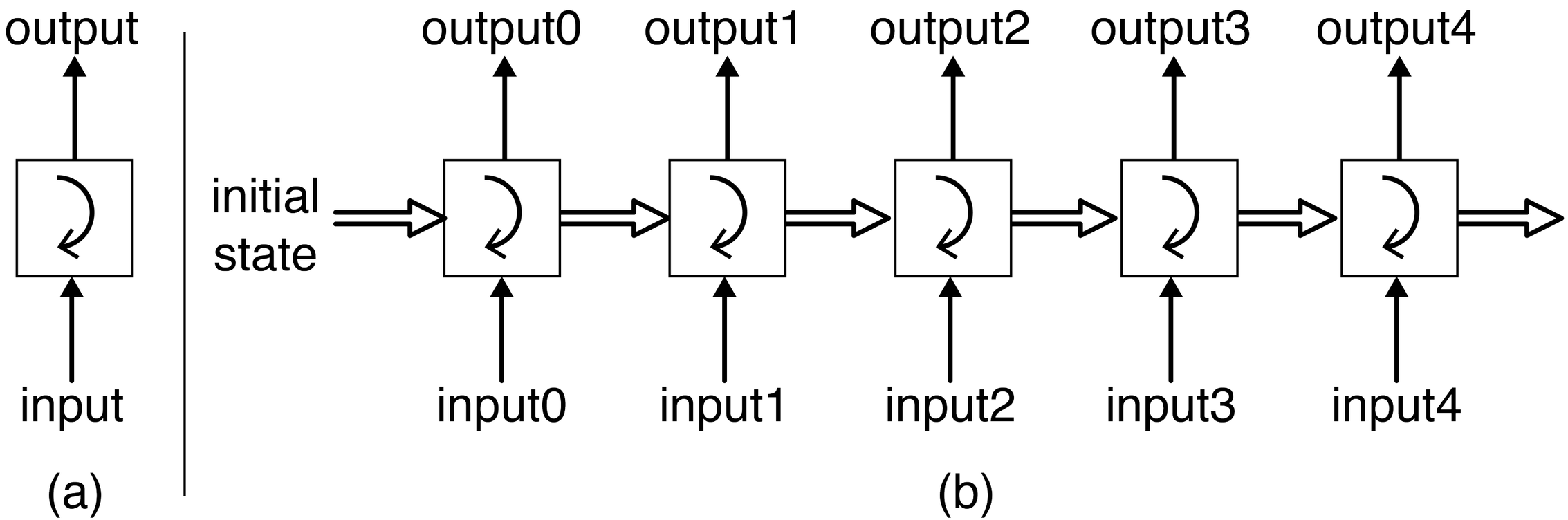
Using Embeddings

```
model = Sequential()
model.add(Embedding(config.vocab_size,
                    config.embedding_dims,
                    input_length=config.maxlen))
model.add(Conv1D(config.filters,
                 config.kernel_size,
                 padding='valid',
                 activation='relu'))
model.add(Flatten())
model.add(Dense(config.hidden_dims, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

#OR

model.add(Embedding(config.vocab_size, 100,
input_length=config.maxlen, weights=[embedding_matrix], trainable=False))
model.add(LSTM(config.hidden_dims, activation="sigmoid"))
```

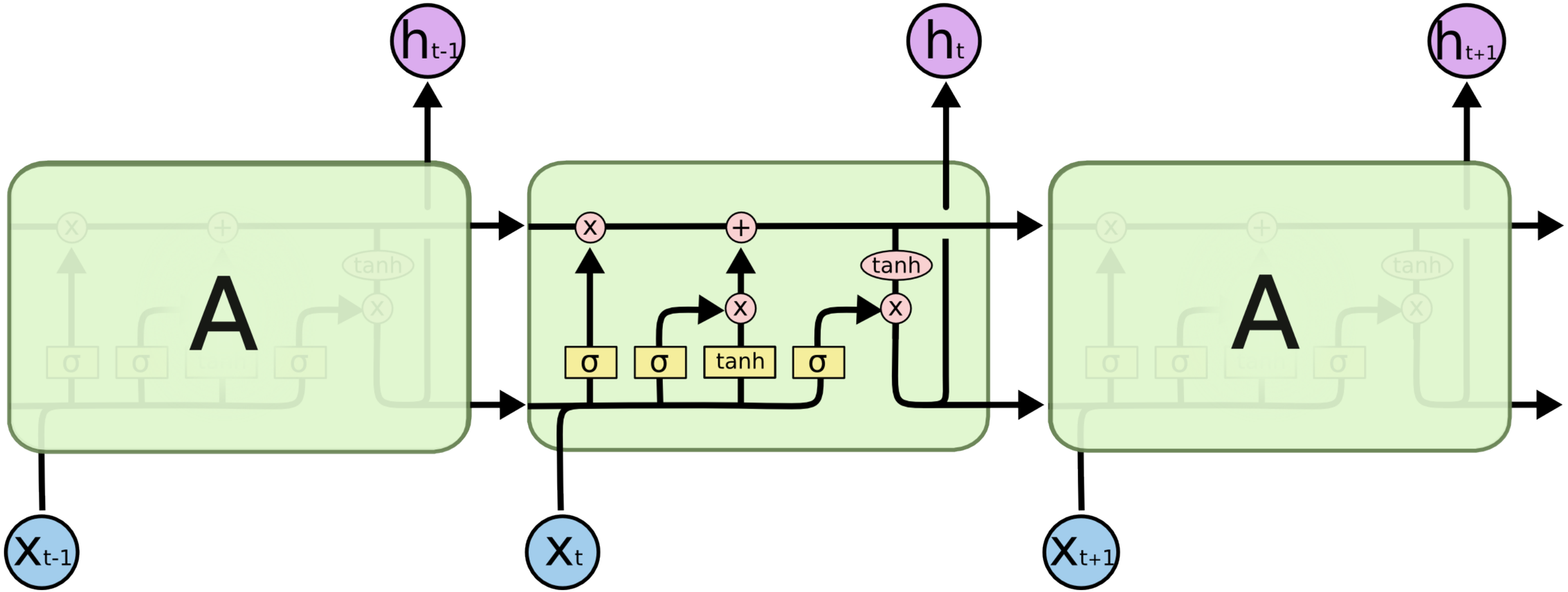
Recurrent Neural networks

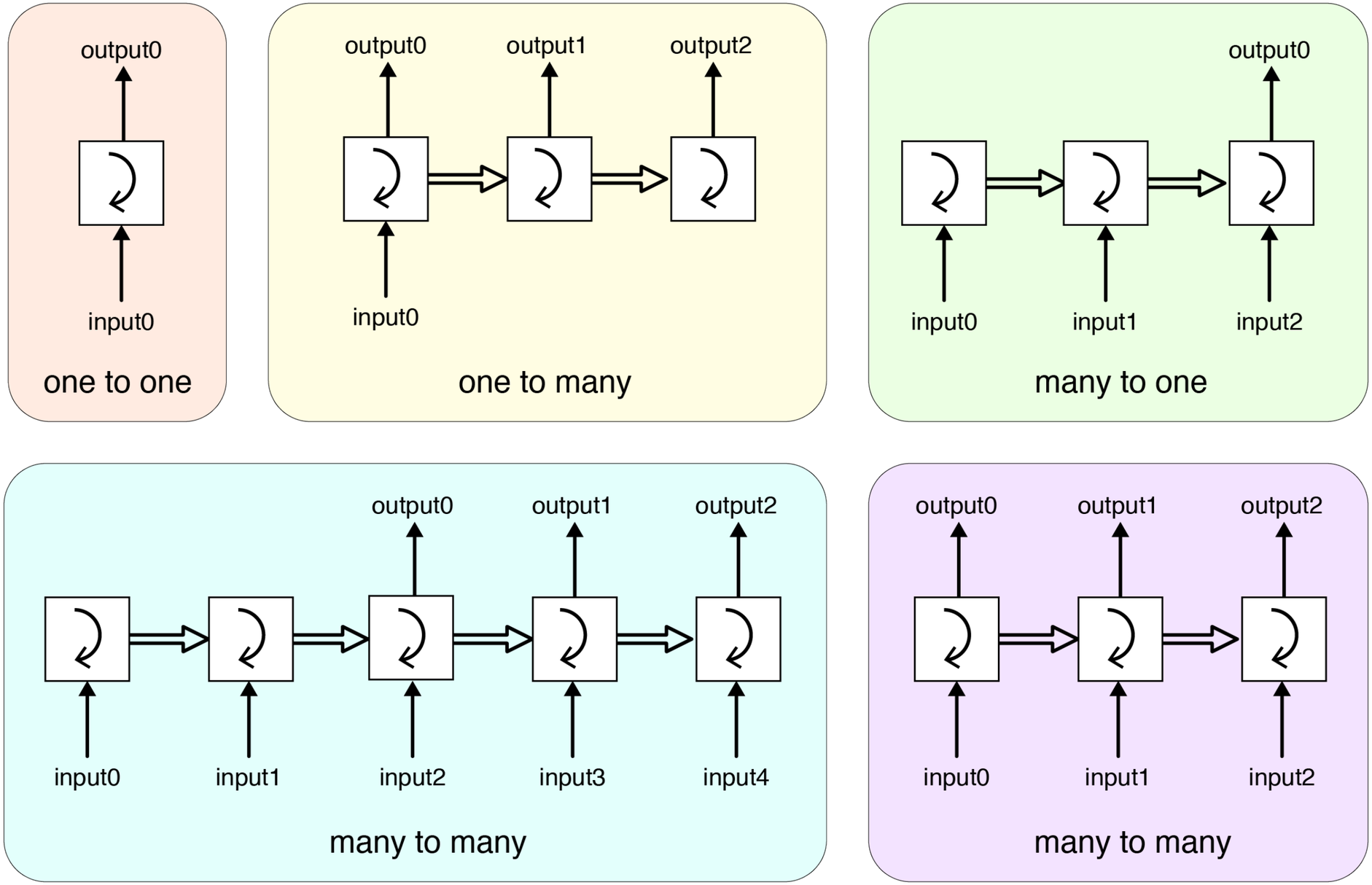


Unrolled...

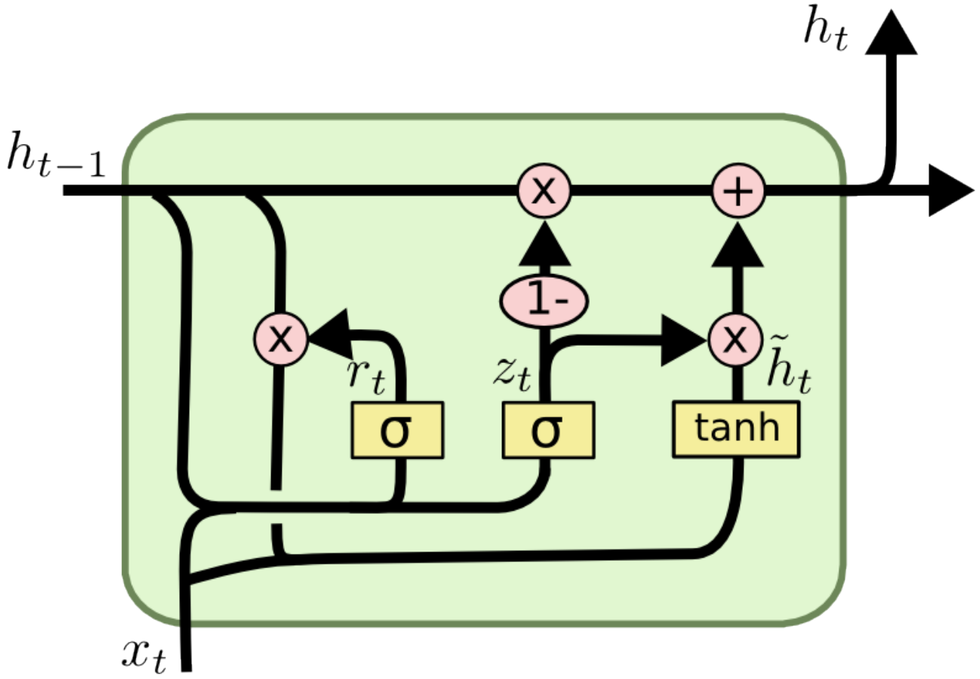

output
input
(a)
initial
state
output0
output1
output2
output3
output4
input0
input1
input2
input3
input4
(b)

LSTM for long term memory (vanishing gradients)




output0
input0
one to one
output0
output1
output2
input0
one to many
output0
input0
input1
input2
many to one
output0
output1
output2
input0
input1
input2
input3
input4
many to many
output0
output1
output2
input0
input1
input2
many to many

GRU



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

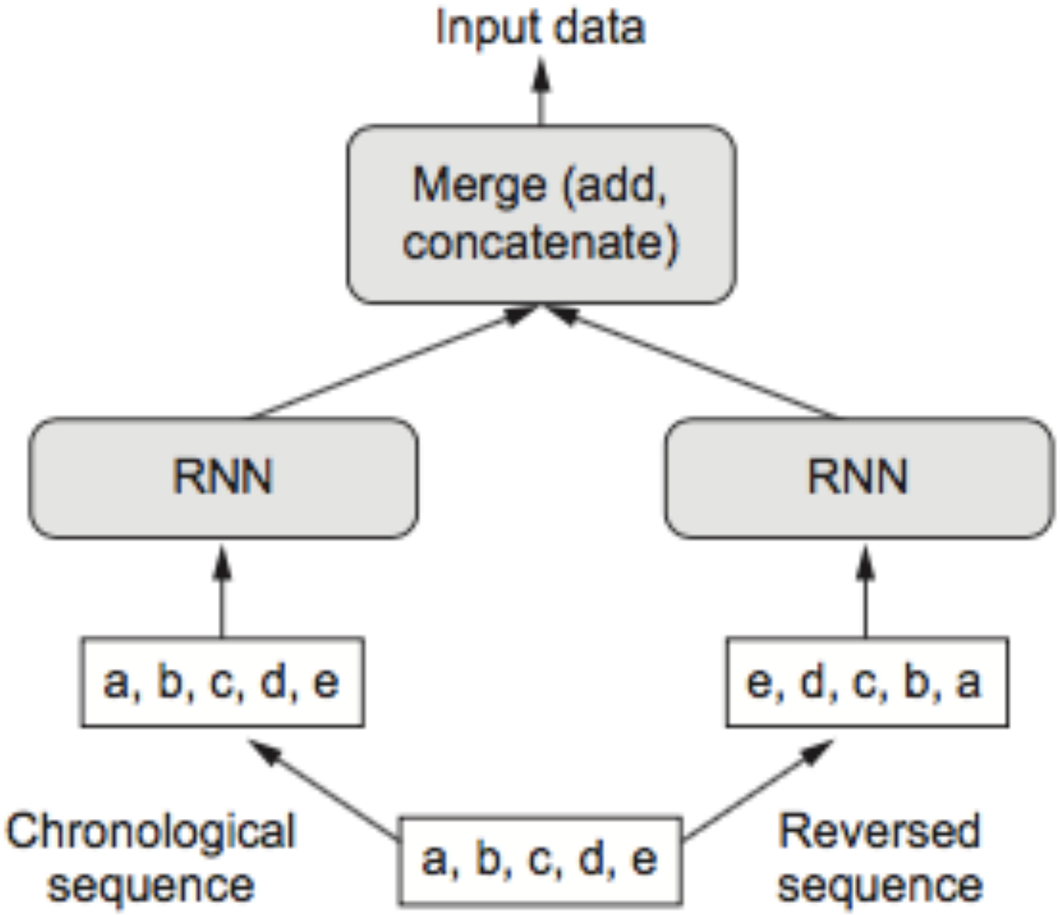
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

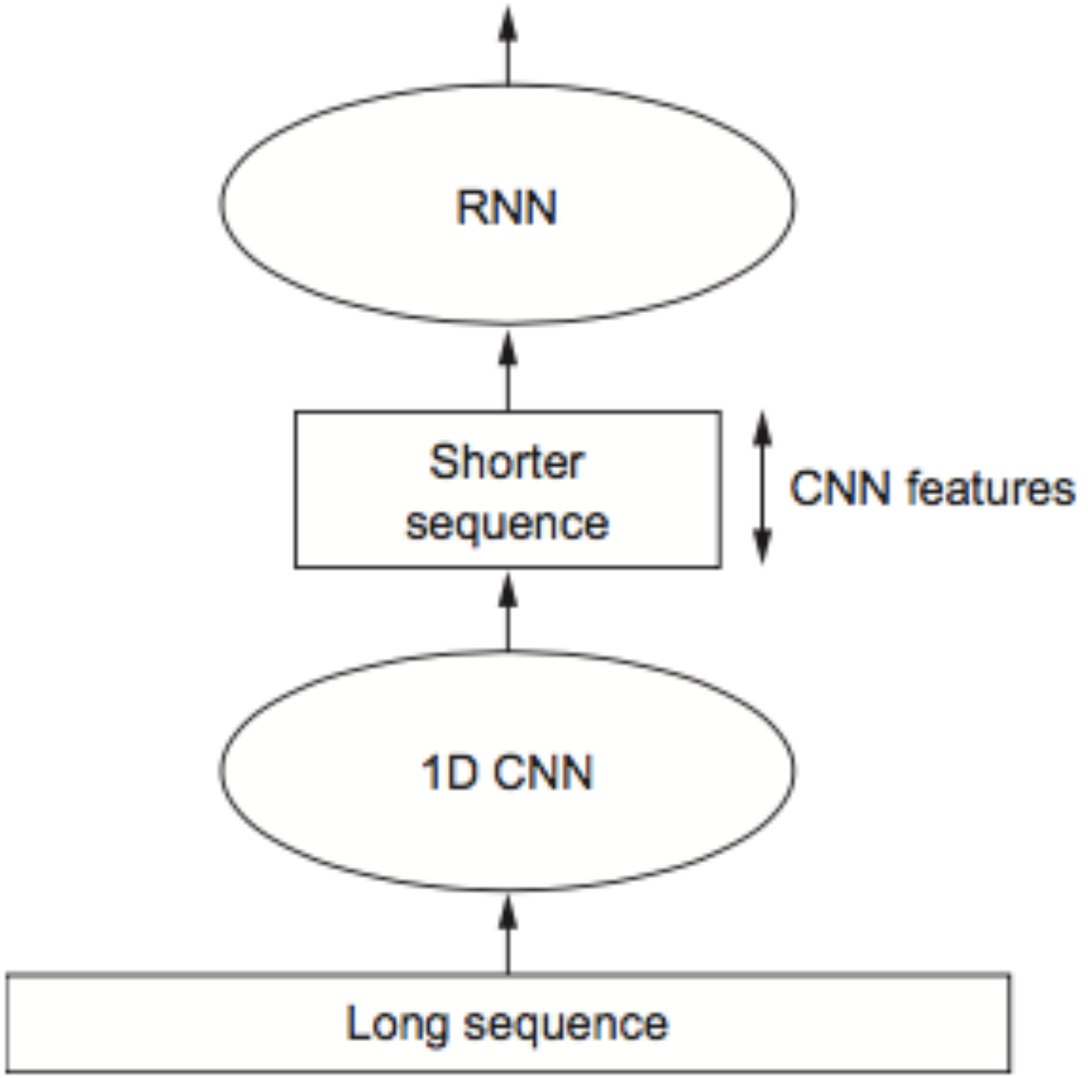
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

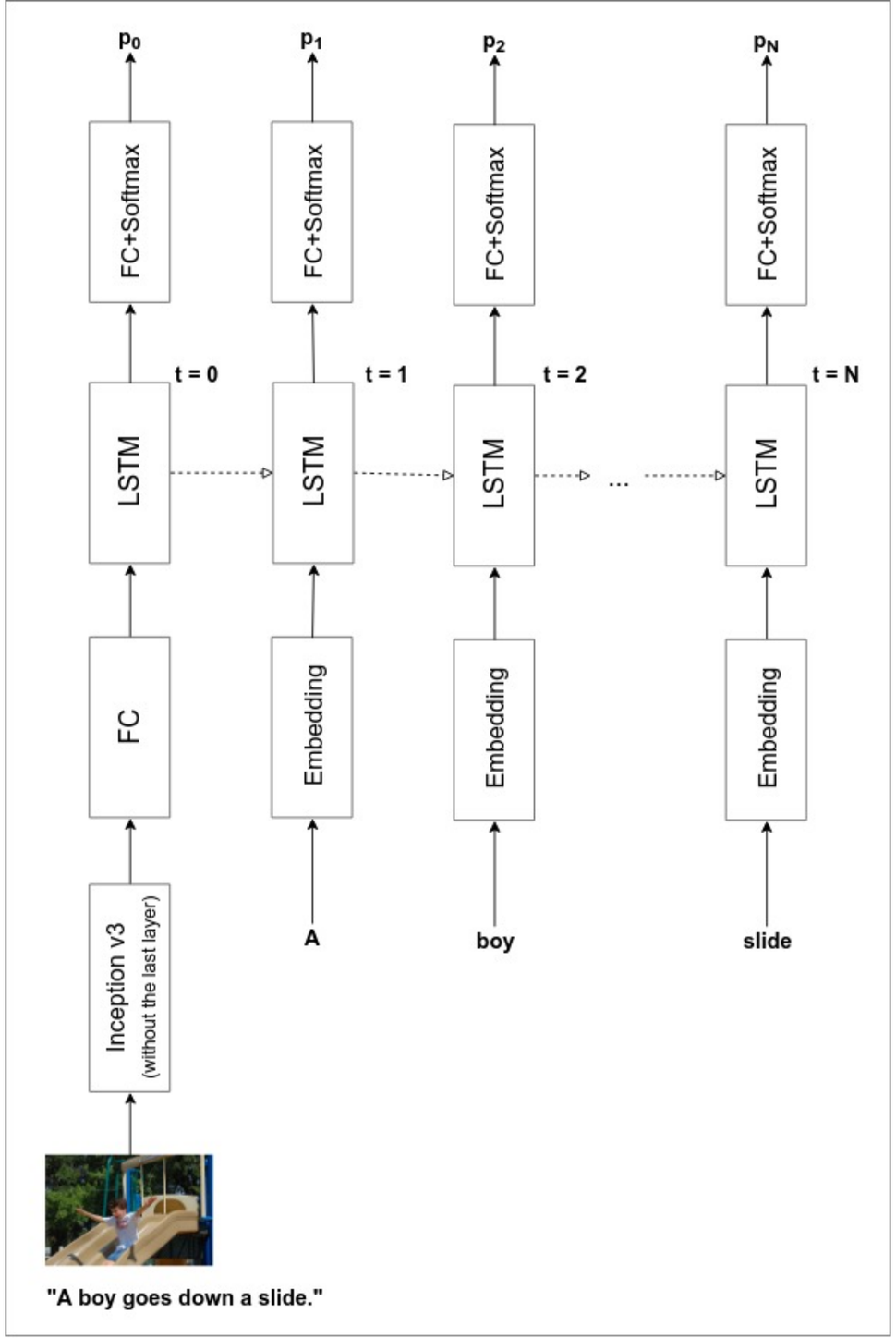
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Other architectures

- cnn on embedding
- cnn-lstm
- stacked deep lstm
- bi-directional lstm
- cnn feeding into part of lstm (captioning)


Input data
Merge (add, concatenate)
RNN
RNN
a, b, c, d, e
e, d, c, b, a
Chronological sequence
a, b, c, d, e
Reversed sequence


RNN
Shorter sequence
CNN features
1D CNN
Long sequence

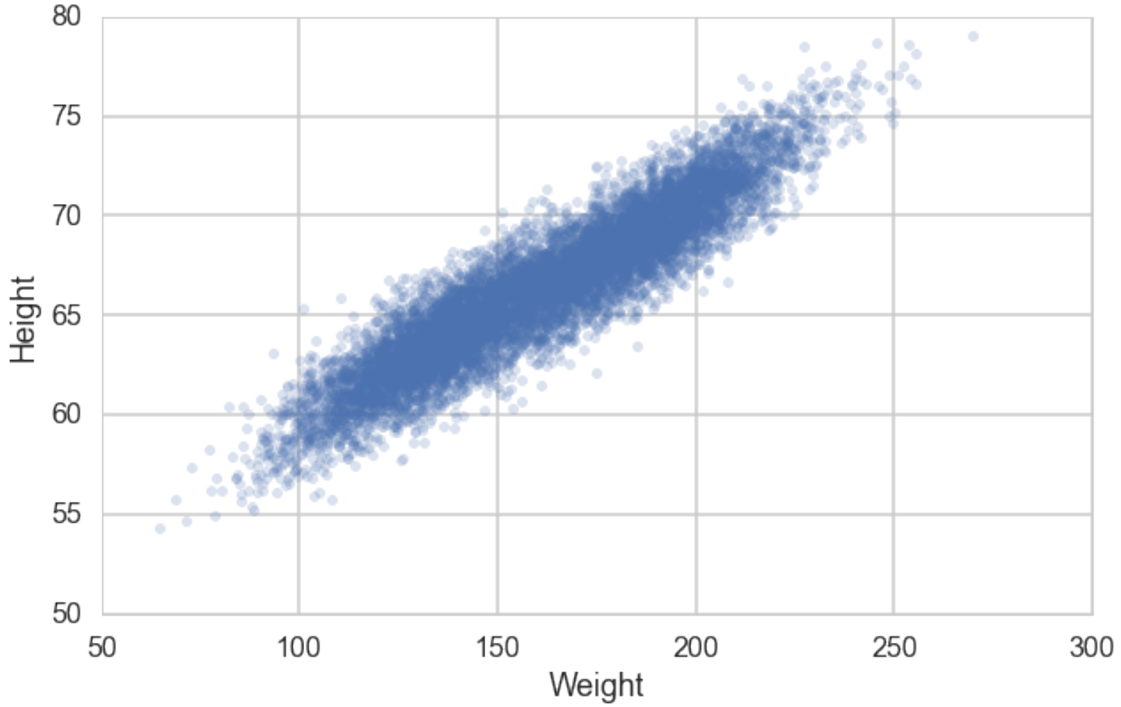


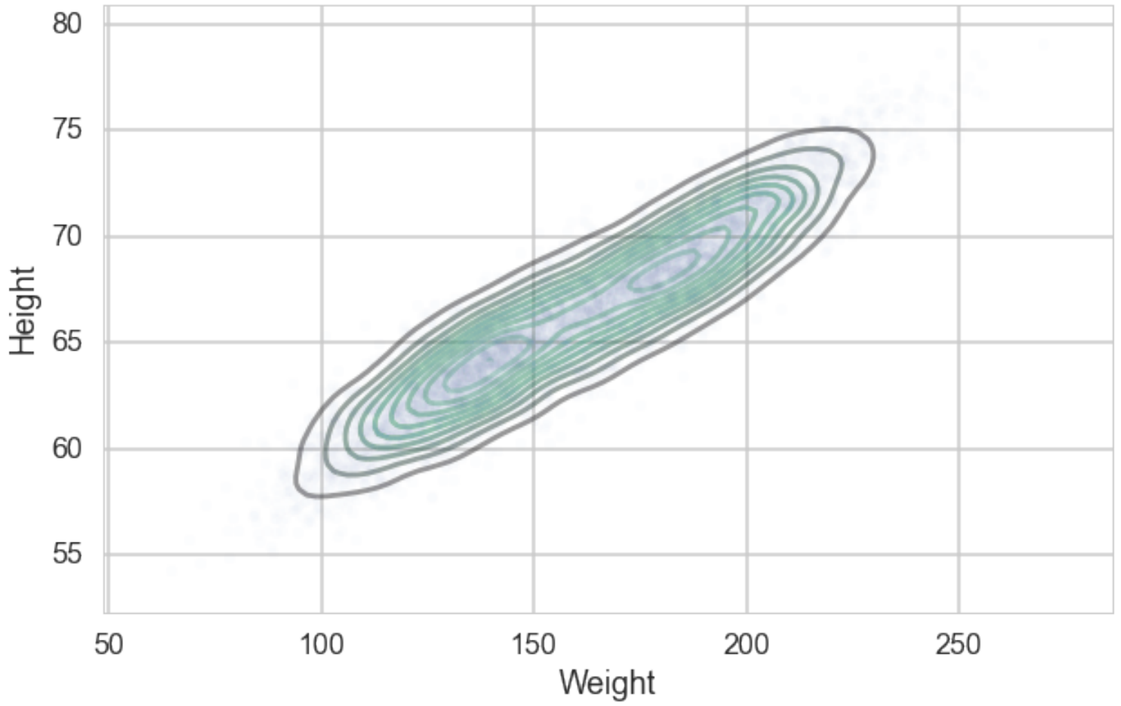
Captioning

What other architectures could we use?

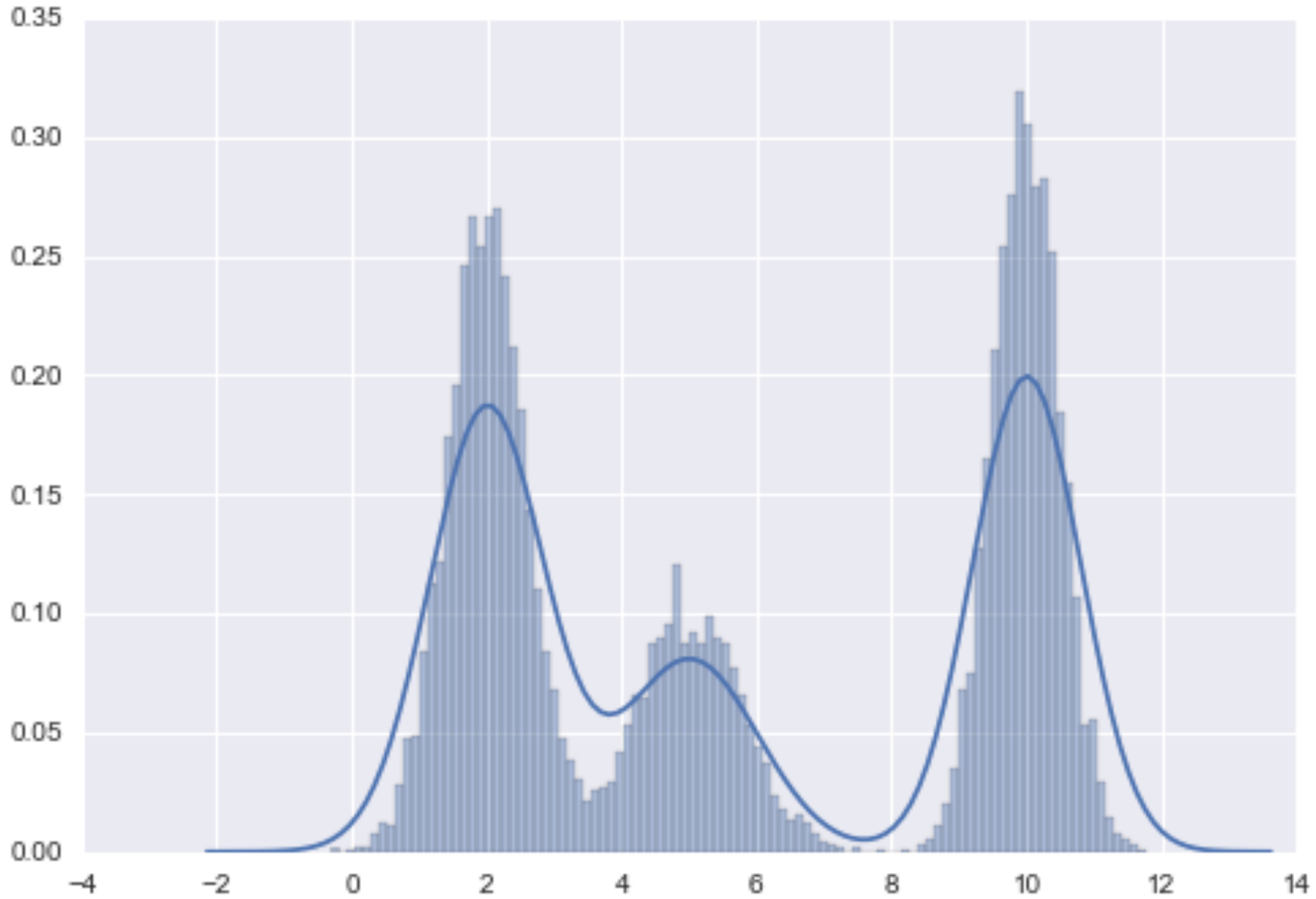
Generative Modeling

p(x)


80
75
70
65
60
55
50
Height
50
100
150
200
250
300
Weight


80
75
70
65
60
55
Height
50
100
150
200
250
Weight

Big Question: Are these classes?



Concrete Formulation of unsupervised learning

$$l(x|\lambda,\mu,\Sigma) = \sum_{i=1}^{m} \log p(x_i|\lambda,\mu,\Sigma)$$

$$= \sum_{i=1}^{m} \log \sum_{z} p(x_i|z_i,\mu,\Sigma)\, p(z_i|\lambda)$$

Not Solvable analytically!

Supervised vs Unsupervised Learning

In **Supervised Learning**, Latent Variables $\mathbf{z}$ are observed.

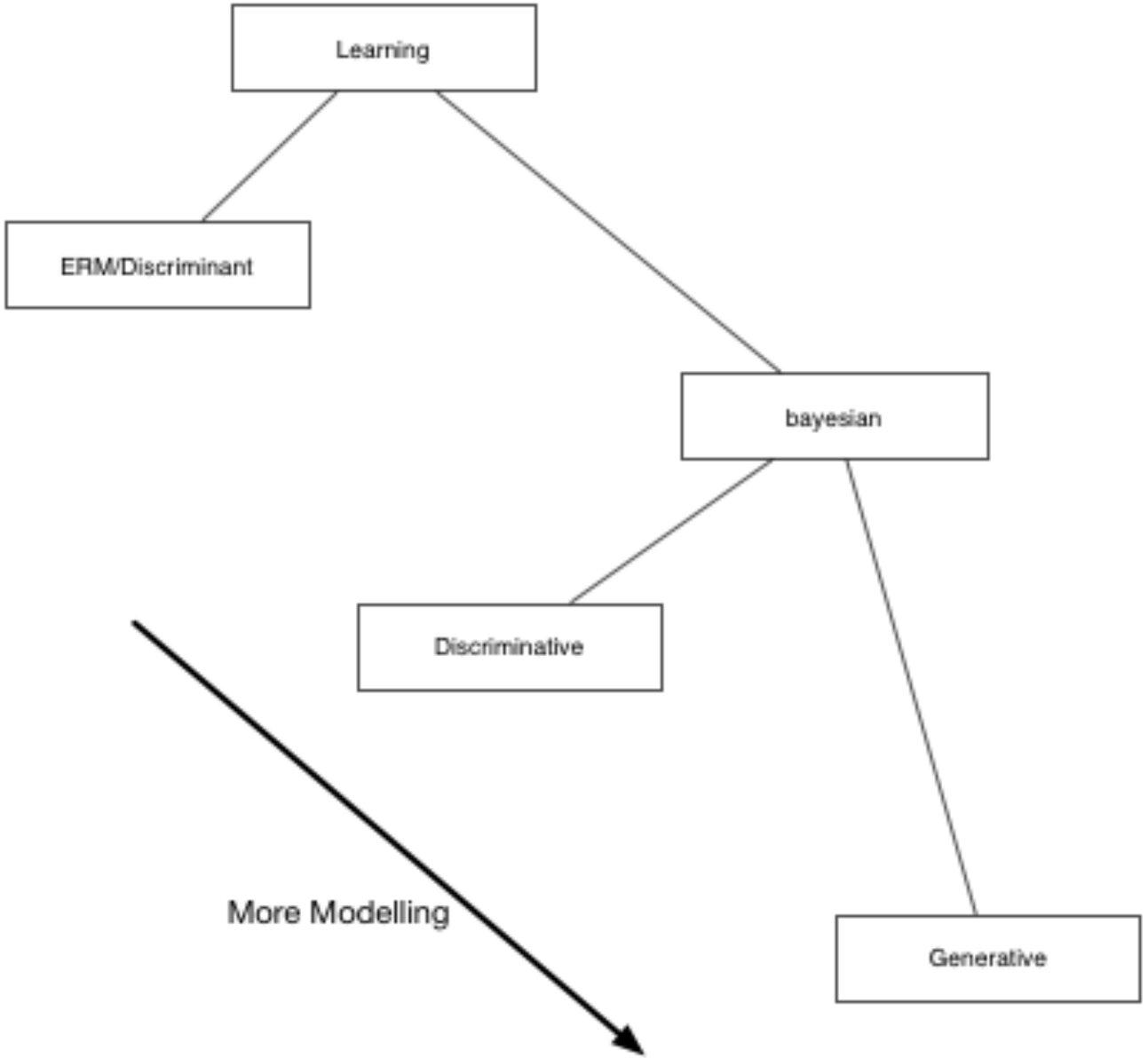
In other words, we can write the full-data likelihood $p(\mathbf{x}, \mathbf{z})$

In **Unsupervised Learning**, Latent Variables $\mathbf{z}$ are hidden.

We can only write the observed data likelihood:

$$p(\mathbf{x}) = \sum_z p(\mathbf{x}, \mathbf{z}) = \sum_z p(\mathbf{z}) p(\mathbf{x}|\mathbf{z})$$

COMBINE: Semi supervized learning.


Learning
ERM/Discriminant
bayesian
Discriminative
Generative
More Modelling

In general

Representations are not classes

they are tangled, hierarchical complex things

but learning them

makes all the difference...

From unsupervised learning

- **"Pure" Reinforcement Learning (cherry)**
  - The machine predicts a scalar reward given once in a while.
  - **A few bits for some samples**
- **Supervised Learning (icing)**
  - The machine predicts a category or a few numbers for each input
  - Predicting human-supplied data
  - **10→10,000 bits per sample**
- **Unsupervised/Predictive Learning (cake)**
  - The machine predicts any part of its input for any observed part.
  - Predicts future frames in videos
  - **Millions of bits per sample**

- **(Yes, I know, this picture is slightly offensive to RL folks. But I'll make it up)**

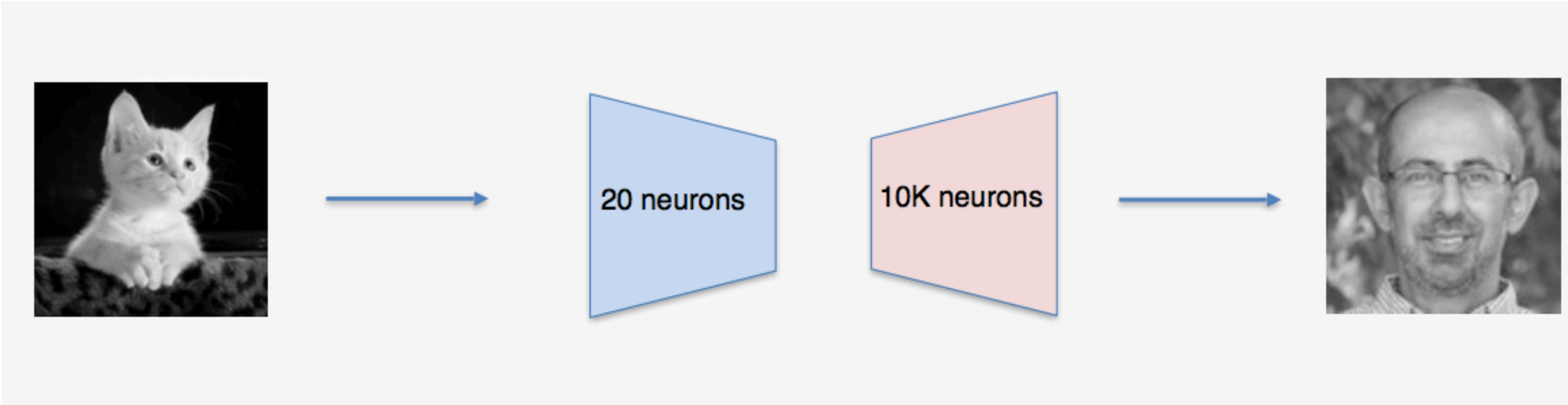
To self-supervized learning

Y. LeCun

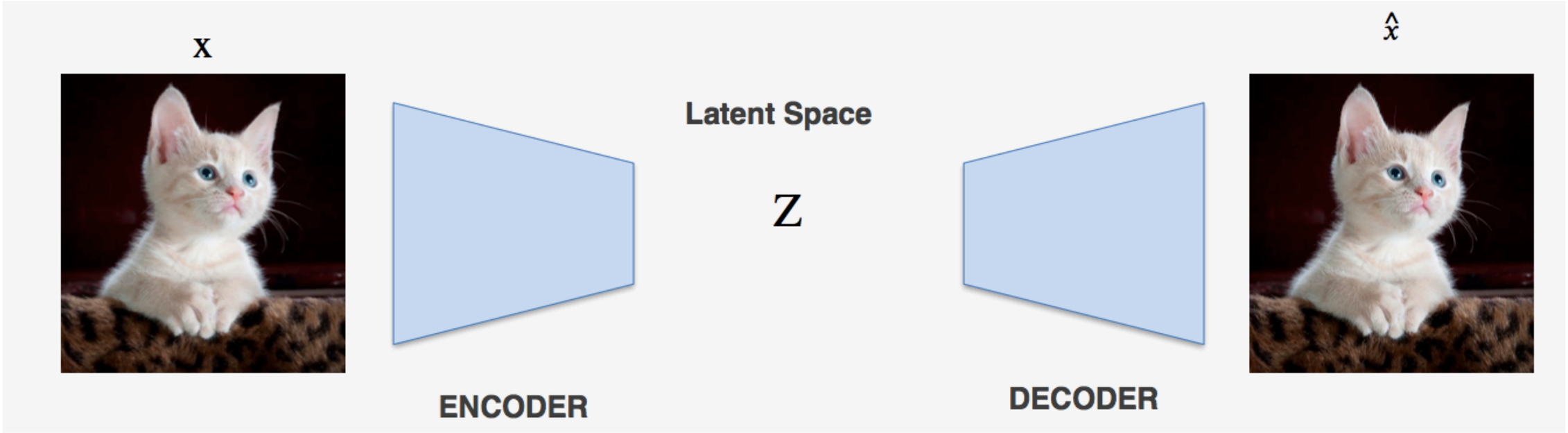
How Much Information is the Machine Given during Learning?

- **"Pure" Reinforcement Learning (cherry)**
  - The machine predicts a scalar reward given once in a while.
  - **A few bits for some samples**
- **Supervised Learning (icing)**
  - The machine predicts a category or a few numbers for each input
  - Predicting human-supplied data
  - **10→10,000 bits per sample**
- **Self-Supervised Learning (cake génoise)**
  - The machine predicts any part of its input for any observed part.
  - Predicts future frames in videos
  - **Millions of bits per sample**

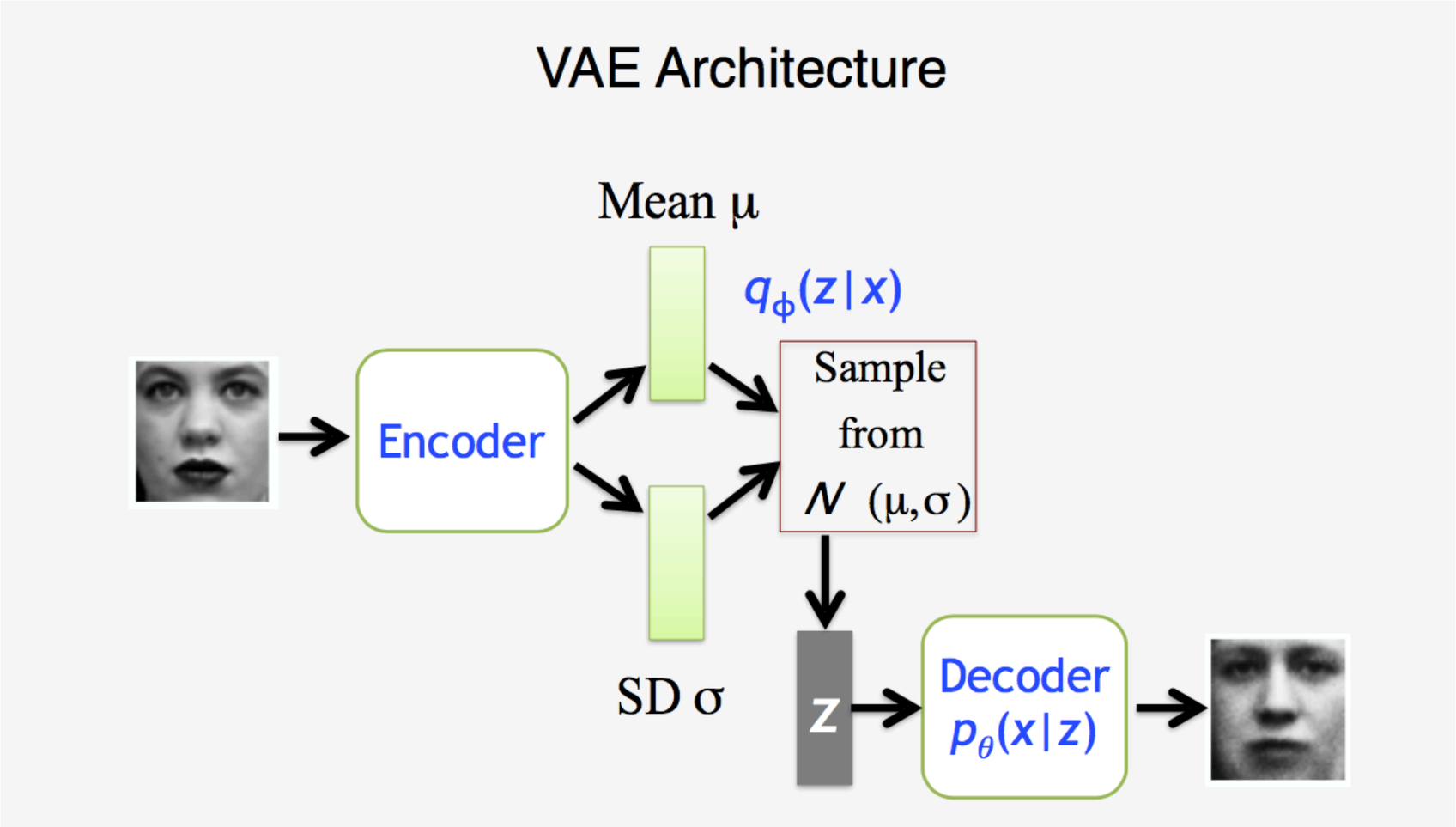
General Encoder-Decoder Architecture



Autoencoder



Variational Autoencoder



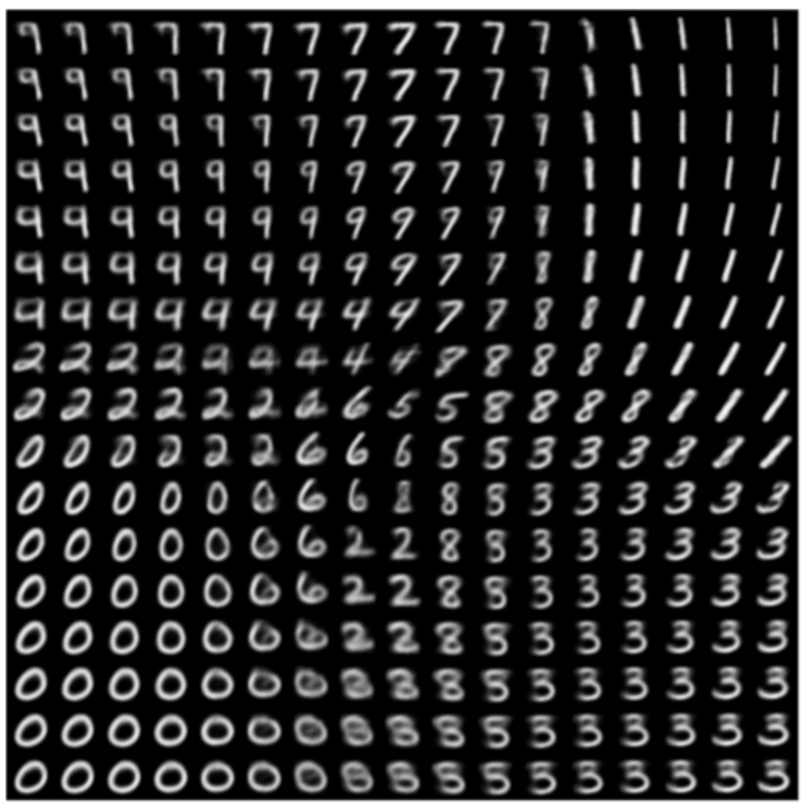
VAE Loss

Reconstruction Loss

Proposal distribution should resemble a Gaussian

$$-\mathbf{E}_{z\sim q_\phi(z|x)}\log\left(p_\theta\left(x|z\right)\right) + KL\left(q_\phi\left(z|x\right)\|p_\theta\left(z\right)\right)$$

Moving from one rep to the other



VAE: Deep Generative Models

- simply not possible to do inference in large models
- inference in neural networks: understanding robustness, etc
- hierarchical neural networks
- Mixture density networks: mixture parameters are fitted using ANNs
- extension to generative semisupervised learning
- variational autoencoders

Where do we go from here?

What should you do?

What should you read? See?

The advanced course

What should you do?

- There is no substitute for coding.
- We are there to help. Please log onto https://discourse.univ.ai, so we can have conversations (use your github id).
- In this next week, choose one of the hackings for the course and work (more) on it.
- we will have a hack every 2 weeks (so 2 a month) until the next basics (next 3 months). we'll discuss on Discourse
- suggest topics if interested!

Come TA for us

- we are looking for TAs for the next basics (hybrid on-site/online, october/november) and the advanced (online, november/december). Ping us if you want to do this!
- TAing is the best way to take your learning to a new level, nothing forces you to understand something like having to explain it.
- you will have the opportunity to develop material, and this will help your understanding
- TA training will be provided

Binge Watching/Reading (watch at 1.5x)

- The first session of fast.ai
- Pattern Recognition and Machine Learning, Bishop
- Deep Learning, Eugene Charniak
- Deep Learning, Andrew Glassner
- Andrew Ng's course is an oldie but goodie. Machine Learning Yearning PDF Document
- Elements of Statistical Learning

The advanced course

- more applications: super-resolution, image segmentation. etc Unets.
- deep unsupervised learning: GANs, more on autoencoders, autoregressive models, flow models
- Transfer learning in NLP with Bert, Elmo, and ULM-Fit
- model deployment and experimentation
- seq2seq, attention
- bayesian statistics

Get on Discourse!