Lecture Notes for

Machine Learning in Python

⟦👨‍🏫,👨‍💻,🐍,👨‍🔬⟧

Professor Eric Larson

Tensorflow Deep Dive

Lecture Agenda

- Logistics:
  - CS 5/7325 in Spring
  - Grading update
- Agenda:
  - More Introduction to TensorFlow
    - Tensors, Tf.Data
    - Deep APIs
  - Wide and Deep Networks
  - Town Hall

Class Overview, by topic

Table Data Visualization

Numpy, Pandas, Seaborn
Overviews with some in-depth discussion

Dimension Reduction and Image Processing

Scikit-learn, Scikit Image,
Intuition only, Some mathematics

Linear and Logistic Regression

Numpy, Recreate API for Scikit-learn
Detailed mathematics for simple optimization
intuition for advanced optimization

Neural Networks and Back Prop.

Numpy
Detailed mathematics for NN operations

Wide and Deep Networks

Convolutional Networks

Sequential Networks

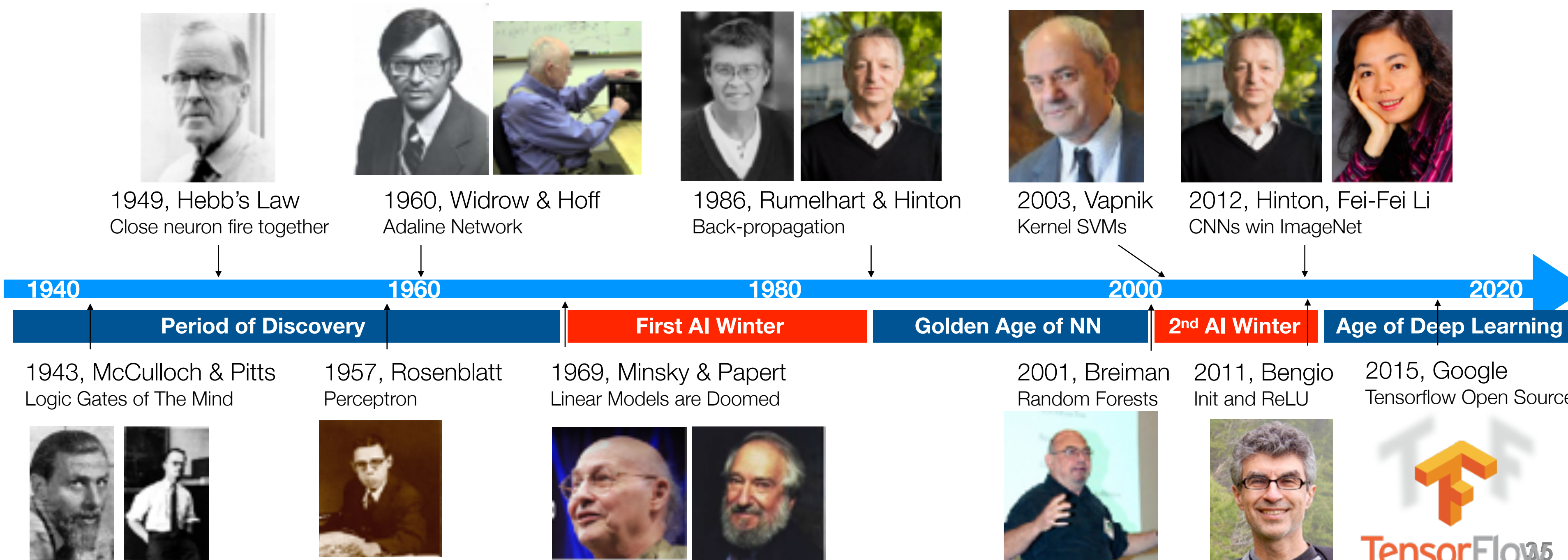
Keras, Tensorflow
Intuition, Detailed implement.

Ethics in Language Models

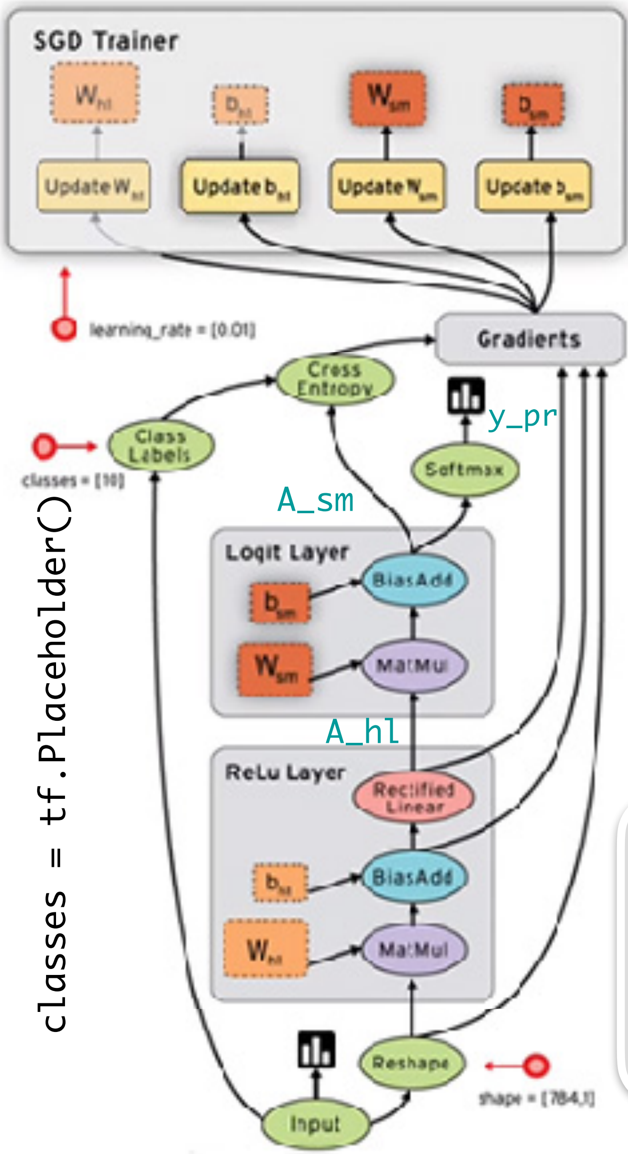
ConceptNet
Case studies

Last Time

- Up to this point: back propagation saved AI winter for NN (Hinton and others!)
- 80's, 90's, 2000's: convolutional networks for image processing start to get deeper
    - but back propagation no longer does great job at training them
- SVMs and Random Forests gain traction…
    - The second AI winter begins, research in NN plummets
- 2004: Hinton secures funding from CIFAR in 2004 Hinton rebrands: Deep Learning
- 2006: Auto-encoding and Restricted Boltzmann Machines
- 2007: Deep networks are more efficient when pre-trained
- 2009: GPUs decrease training time by 70 fold…
- 2010: Hinton's students go to internships with Microsoft, Google, and IBM, making their speech recognition systems faster, more accurate and deployed in only 3 months…
- 2012: Hinton Lab, Google, IBM, and Microsoft jointly publish paper, popularity sky-rockets for deep learning methods
- 2011-2013: Ng and Google run unsupervised feature creation on YouTube videos (becomes computer vision benchmark)
- 2012+: Pre-training is not actually needed, just solutions for vanishing gradients (like ReLU, SiLU, initializations, more data, GPUs)



Last Time




```
Input = tf.Placeholder() # size is 28x28
Input = tf.Reshape(Input, [784,1])
classes = tf.Placeholder()

W_sm = tf.Variable(…)
b_sm = tf.Variable(…)
W_hl = tf.Variable(…)
b_hl = tf.Variable(…)
```

```
trainable_variables =
        [W_sm,b_sm,W_hl,b_hl]
```

```
def model_forward(Input):
    A_hl = tf.relu( tf.matmul(Input,W_hl) + b_hl )
    A_sm = tf.matmul(A_hl,W_sm) + b_sm
    return A_sm
```

```
y_pr = tf.softmax(A_sm)
loss = tf.sparse_softmax_cross_entropy_with_logits
```

```
opt = tf.train.SGDOptimizer(learning_rate=0.01)
```

```
for features, labels in train_data:
    with tf.GradientTape( ) as tape:
        yhat = model_forward(features)
        loss_val = loss(labels, yhat)
    grads = tape.gradient(loss_val, trainable_variables)
    opt.apply_gradients(zip(grads, trainable_variables))
```

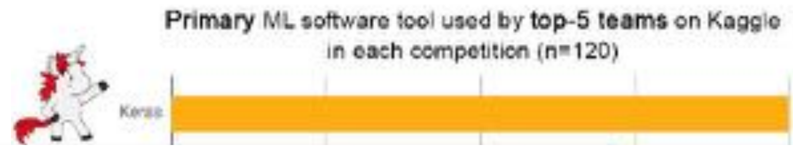
Self Test

- The computation graph in tensorflow:
  - A. Can run one operation at a time
  - B. Can be “compiled” for efficiency
  - C. Has one input path (e.g., features) and one output path (e.g., predictions)
  - D. All of the Above

Using Keras and Tensorflow

Keras Programming Interfaces

```
from tensorflow import keras
```



- **Keras Sequential API**
  - great for simple, feed forward models

```
my_model = Sequential([
        Dense(100, activation='sigmoid'),
        Dense(10)
     ])
```

- **Keras Functional API**
  - build models through series of nested functions
  - each “function” represents an operation in the NN

```
x_in = Input(shape=(1,))

x = Dense(100, activation='sigmoid')(x_in)
x = Dense(10)(x)

my_model = Model(inputs = x_in,
                 outputs = x )
```

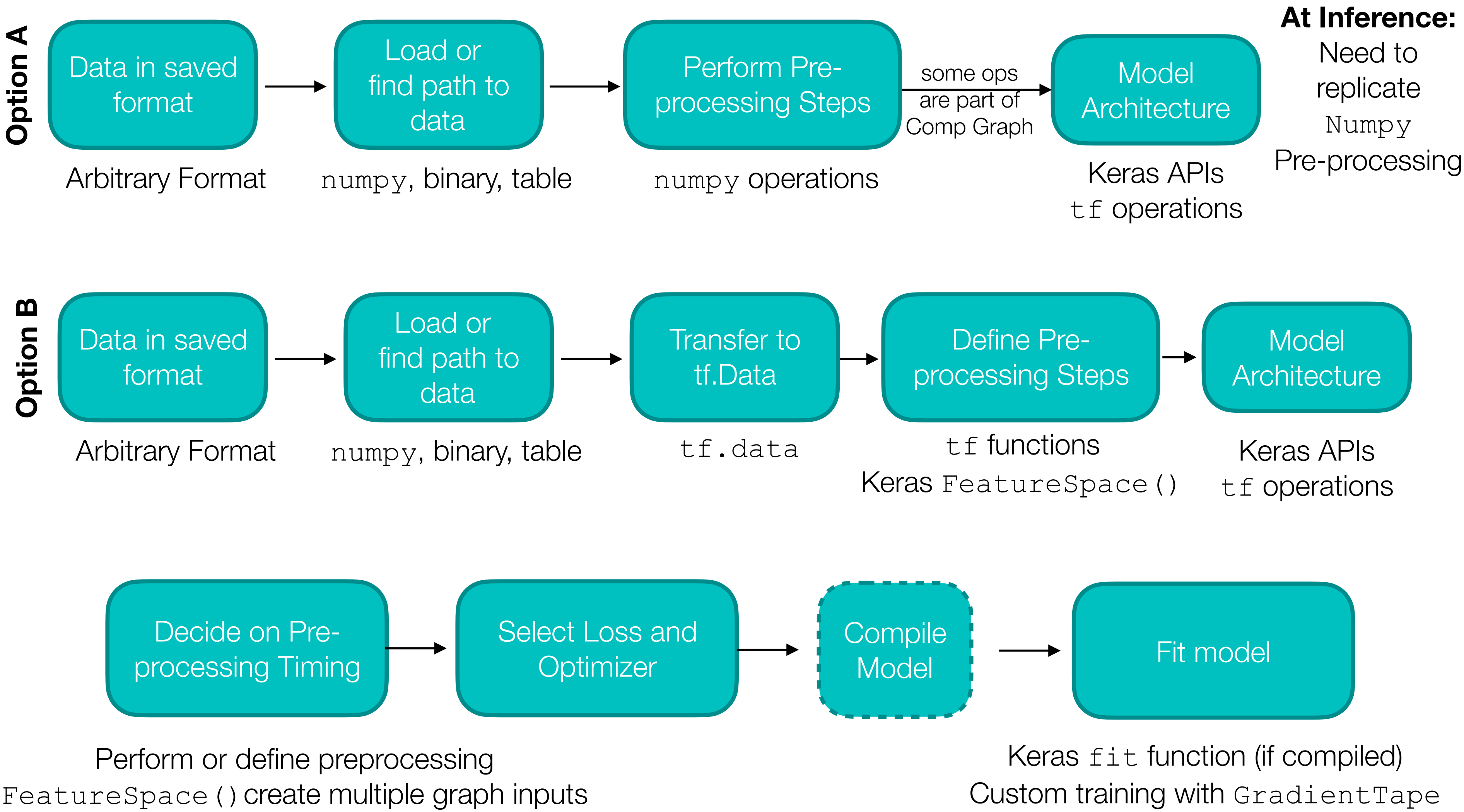
- **Keras Classes (Inheritance)**
  - good for more advanced functionality

```
class CustomModel(keras.Model):
  def __init__(self, num_classes=10):
    super().__init__()
    self.dense1 = Dense(100, activation='sigmoid')
    self.output_layer = Dense(num_classes)

  def call(self, inputs):
    x = self.dense1(inputs)
    return self.output_layer(x)

my_model = CustomModel(10)
```

Using Keras and Tensorflow



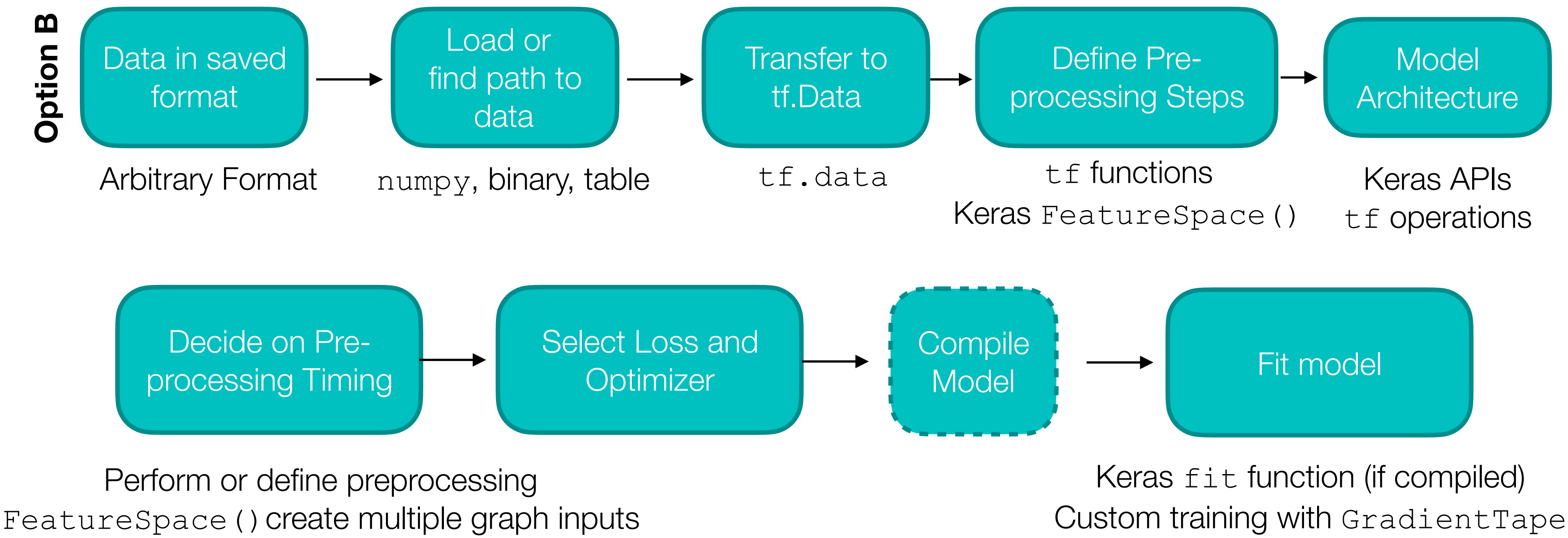
- **Computation Graph:**
  - Compiled models use optimized CG
  - GradientTape uses Eager execution by Default

Revisiting the MLP with Keras

`10a. Keras Wide and Deep as TFData.ipynb`

Make me slow down if I go too fast!!

Using Keras and Tensorflow

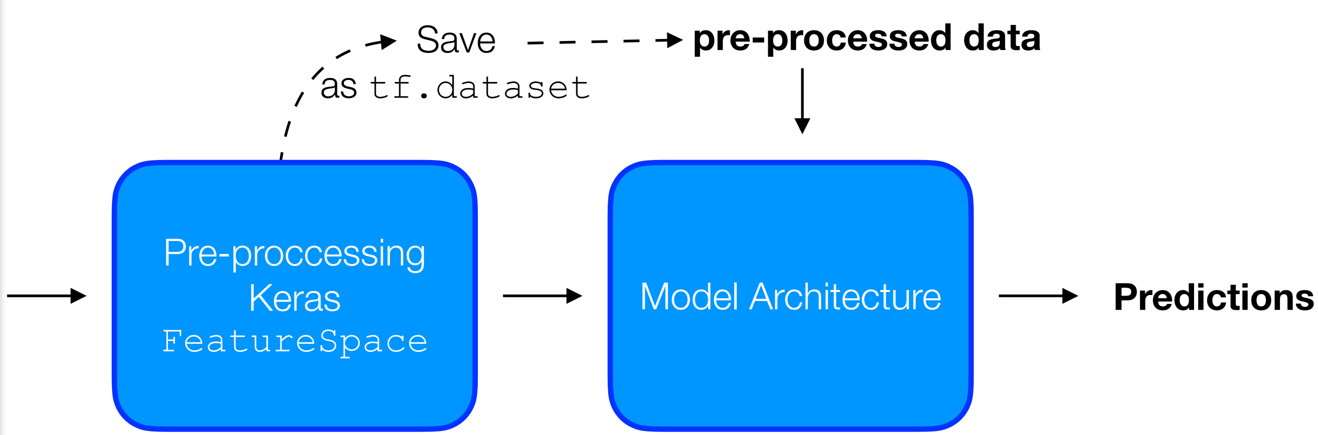


FeatureSpace Methods:

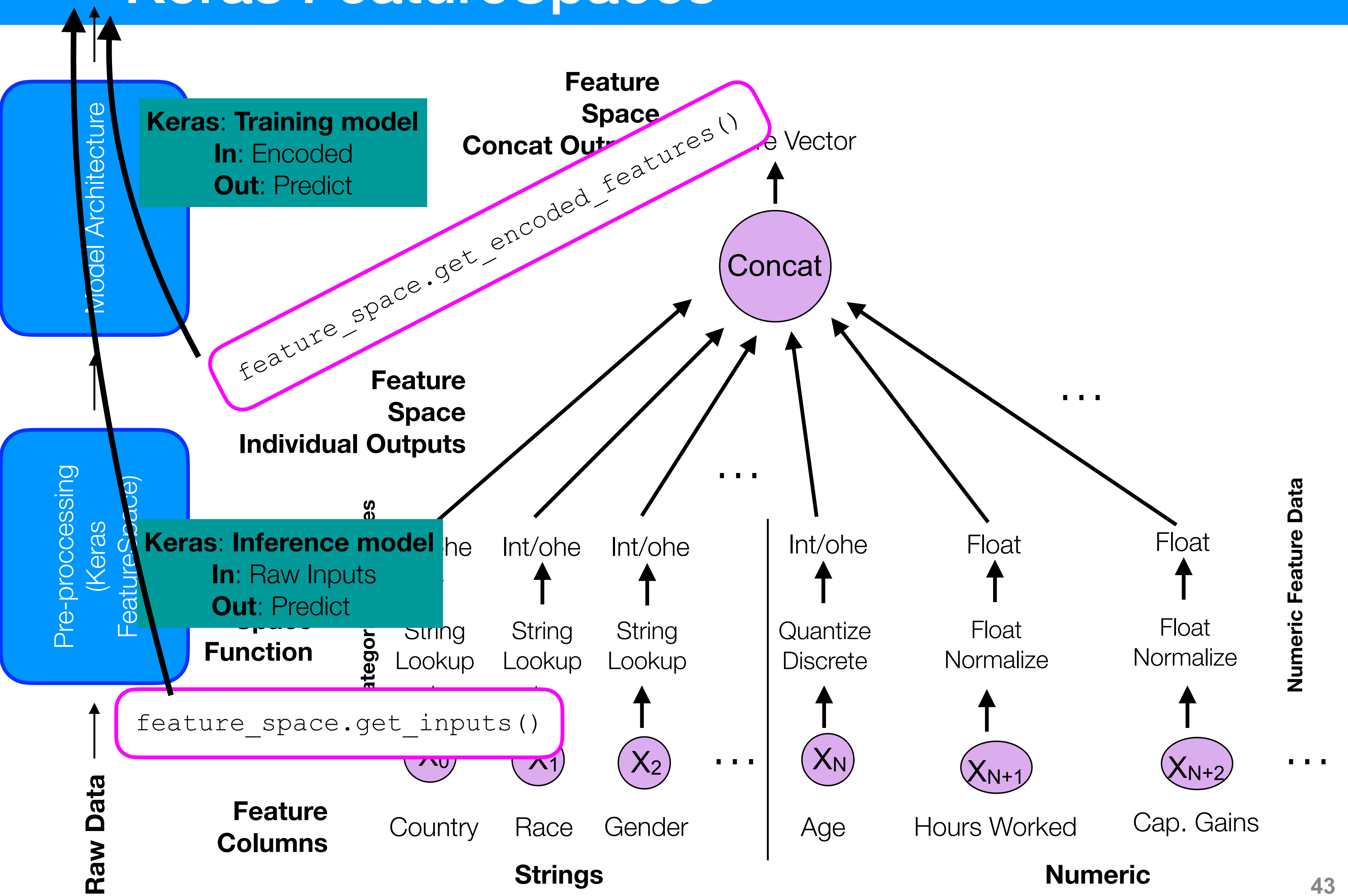
```
integer_categorical
integer_hashed

string_categorical
string_hashed

float
float_discretized
float_normalized
float_rescaled
```



Keras FeatureSpaces



Demo

Setting up Feature Spaces

`10a. Keras Wide and Deep as TFData.ipynb`

Categorical Feature Embeddings

- One hot encoded data can be made dense through a matrix multiplication, $\mathbf{a} = \mathbf{W}^{(emb)} \cdot \mathbf{x}_{OHE}$

Trainable Matrix, $\mathbf{W}^{(emb)}$, in Network

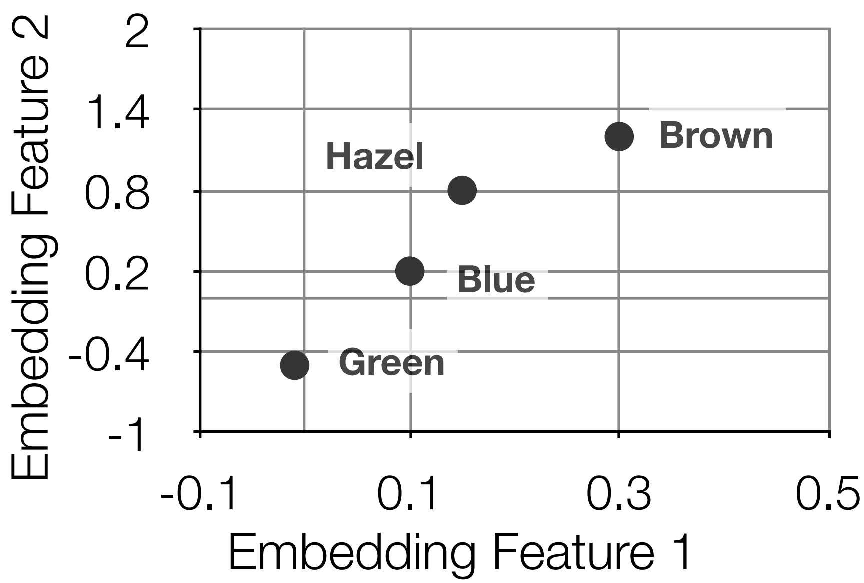
Reduced Dimensions

0.1	0.3	-0.01	0.15
0.2	1.2	-0.5	0.8

Num Categories

Eye Color

$$\cdot \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \begin{array}{l} \text{Blue} \rightarrow 0 \\ \text{Brown} \rightarrow 1 \\ \text{Green} \rightarrow 2 \\ \text{Hazel} \rightarrow 3 \end{array}$$



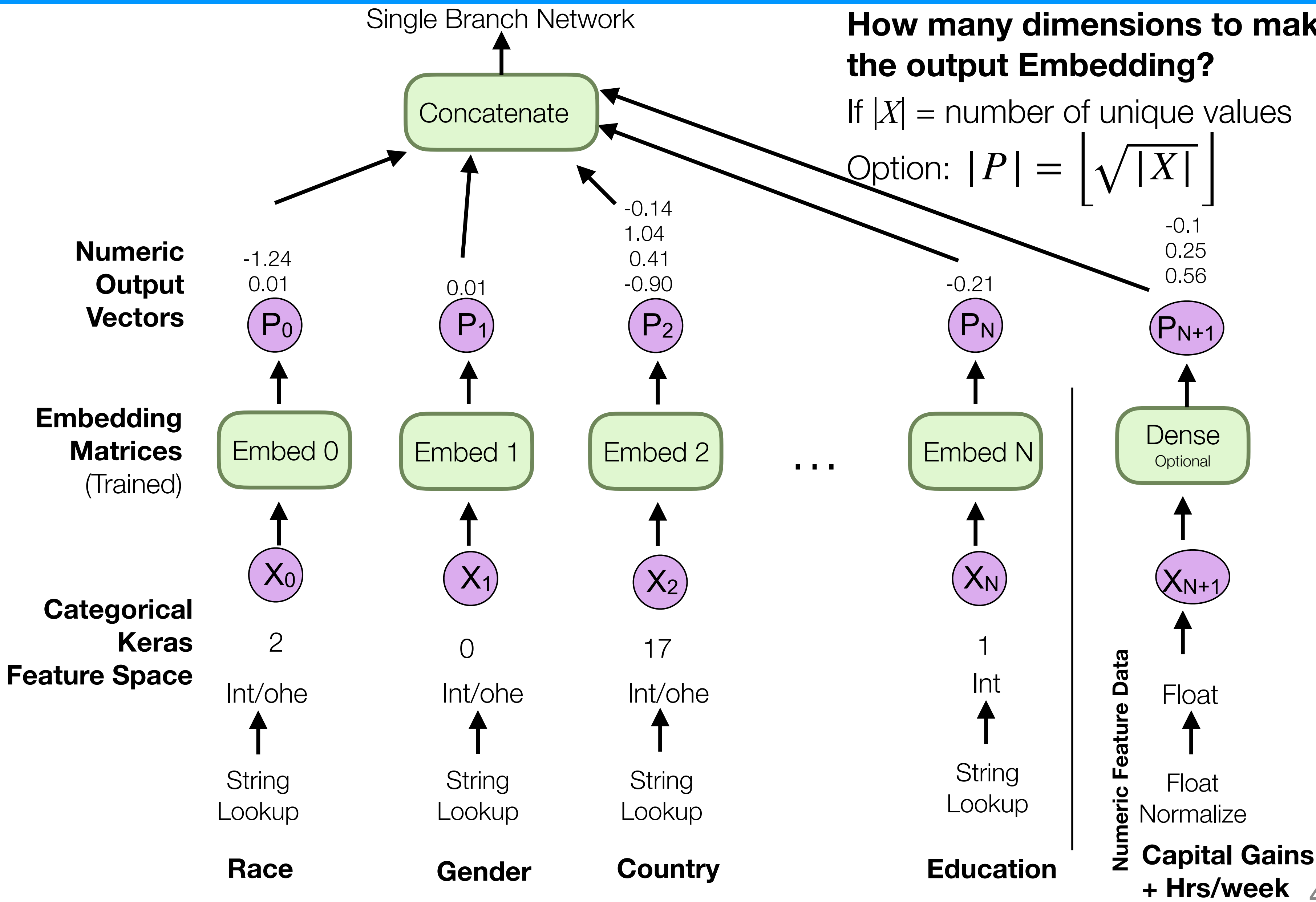
Computationally: there is no need to one hot encode eye color, we can just use the integer to index into the **embedding matrix**

Using Embeddings in Keras

How many dimensions to make the output Embedding?

If $|X|$ = number of unique values

Option: $|P| = \left\lfloor \sqrt{|X|} \right\rfloor$



Demo

Adding Embedding Branches

`10a. Keras Wide and Deep as TFData.ipynb`