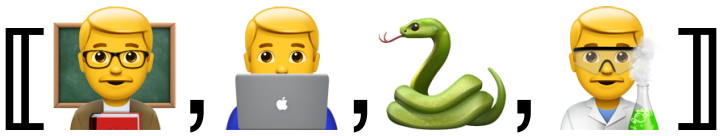
Lecture Notes for

# Machine Learning in Python

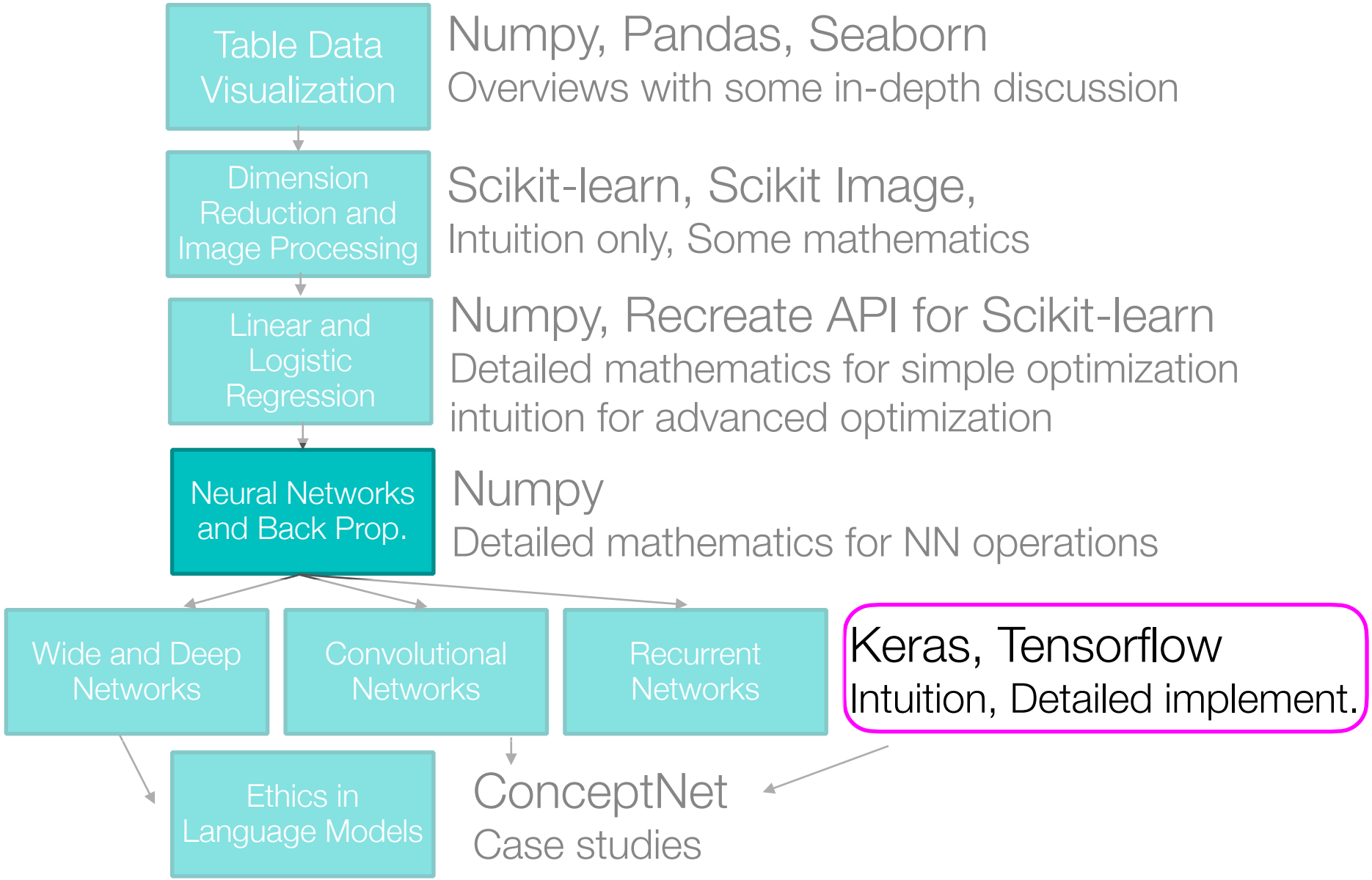


## A History of Deep Learning And Introduction to Toolkits

# Logistics and Agenda

- Logistics
  - Grading update
- Agenda
  - “Deep Learning” History
  - Deep Learning toolkits

# Class Overview, by topic

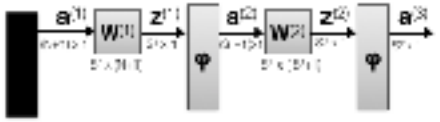


# Last time:

## Back propagation summary

$$J(\mathbf{W}) = \sum_k^M (\mathbf{y}^{(k)} - \mathbf{a}^{(L)})^2$$

$$w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{z}^{(l)}} a_j^{(l)}$$



1. Forward propagate to get **z**, **a** for all layers
2. Get final layer gradient
3. Update back propagation variables
4. Update each $\mathbf{W}^{(l)}$

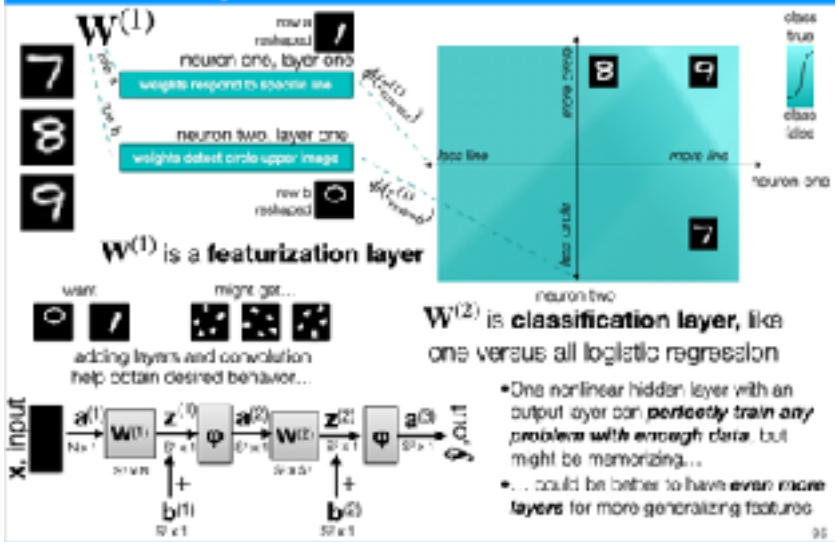
for each $\mathbf{y}^{(k)}$

$$\frac{\partial J(\mathbf{W})}{\partial \mathbf{z}^{(2)}} = -2(\mathbf{y}^{(k)} - \mathbf{a}^{(2)}) * \mathbf{a}^{(2)} * (1 - \mathbf{a}^{(2)})$$

$$\frac{\partial J(\mathbf{W})}{\partial \mathbf{z}^{(l)}} = \text{diag}[\mathbf{a}^{(l+1)} * (1 - \mathbf{a}^{(l+1)})] \cdot \mathbf{W}^{(l+1)} \frac{\partial J(\mathbf{W})}{\partial \mathbf{z}^{(l+1)}}$$

$$\mathbf{W}^{(l)} \leftarrow \mathbf{W}^{(l)} - \eta \frac{\partial J(\mathbf{W}^{(l)})}{\partial \mathbf{z}^{(l)}} \cdot \mathbf{a}^{(l)}$$

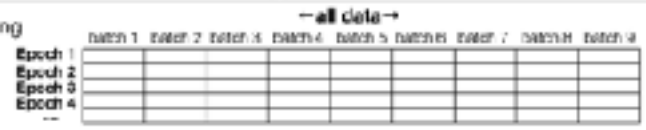
## Universality




## Review

$$\mathbf{W}_{k+1} = \mathbf{W}_k - \rho_k$$

- Cross entropy: $\mathbf{Y}^{(2)} \equiv \mathbf{A}^{(2)} - \mathbf{Y}$ now final layer update
- Momentum: $\rho_k = \alpha \nabla J(\mathbf{W}_k) + \beta \nabla J(\mathbf{W}_{k-1})$
- Nesterov's Momentum: $\rho_k = \beta \nabla J\left(\mathbf{W}_k + \alpha \nabla J(\mathbf{W}_{k-1})\right) + \alpha \nabla J(\mathbf{W}_{k-1})$ — step twice
- Mini-batching

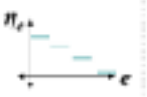



*shuffle ordering each epoch and update W's after each batch*

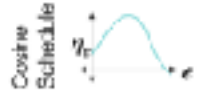
- Learning rate adjustment (eta)

$$\eta_e = \eta_{min} + \frac{1}{2}(\eta_{max} - \eta_{min})\left(1 + \cos\left(\frac{e}{e_{max}}\pi\right)\right)$$

Step step: $\eta_e = \eta_0 \cdot d^{\lfloor \frac{e}{s} \rfloor}$, $s$: epochs between reductions, $0 < d < 1$



Cosine Schedule



## McNemar Example

| Model 1 | Model 2 | Label | Matrix |
|---|---|---|---|
| T-shirt | T-shirt | T-shirt | A |
| Sneaker | T-shirt | Sneaker | B |
| T-shirt | Pullover | Pullover | C |
| Sneaker | Sneaker | Sneaker | A |
| T-shirt | Sneaker | Sneaker | D |
| Pullover | Pullover | T-shirt | D |
| Pullover | T-shirt | Pullover | B |
| Sneaker | Sneaker | Sneaker | A |
| Sneaker | Sneaker | Sneaker | A |

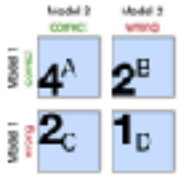
McNemar and Edwards, 1948

$$\chi^2 \approx \frac{(|B - C| - 1)^2}{B + C}$$

$$\chi^2 = \frac{(|2 - 2| - 1)^2}{2 + 2} = 0.25$$

| Confidence | 0.90 | 0.95 | 0.99 |
|---|---|---|---|
| 1 DOF, Critical Value | 2.706 | 3.841 | 6.635 |

https://www.itl.nist.gov/div898/handbook/eda/section3/eda3674.htm




Since 0.25 < 3.841, we cannot reject the null hypothesis. This means **we should not say the models' performance are different** based on the evidence.

# Self Test

- Modern day machine learning scientists have a rigorous background in neural biology or in processes that are biologically plausible.
  - **A. True.** Neural networks are windows into the mind.
  - **B. True.** PhD's in Machine Learning require advanced biology study.
  - **C. True.** They have made bold claims about human intelligence, and therefore must have this expertise.
  - **D. False.** They receive no formal training in this nor do they regularly consult with experts in this field.

# Some History of Deep Learning

# Machine Learning Timeline (Neural Nets)

- Up to this point: back propagation ended first AI winter
- 80's, 90's: neural networks for image processing start to get deeper
  - but back propagation no longer efficient for training
  - Back propagation gradient **stagnates** research—can't train **deeper** networks

- Second AI winter begins, research in NN plummets
- Funding for and accepted papers with Neural Networks asymptotically approaches zero

1949, Hebb's Law
Close neuron fire together

1960, Widrow & Hoff
Adaline Network

1986, Rumelhart & Hinton
Back-propagation

2003, Vapnik
Kernel SVMs


1940 | 1960 | 1980 | 2000 | 2020

Period of Discovery | First AI Winter | Golden Age of NN | 2nd AI Winter | Age of Deep Learning

1943, McCulloch & Pitts
Logic Gates of The Mind

1957, Rosenblatt
Perceptron

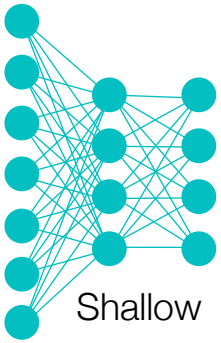
1969, Minsky & Papert
Linear Models are Doomed

2001, Breiman
Random Forests

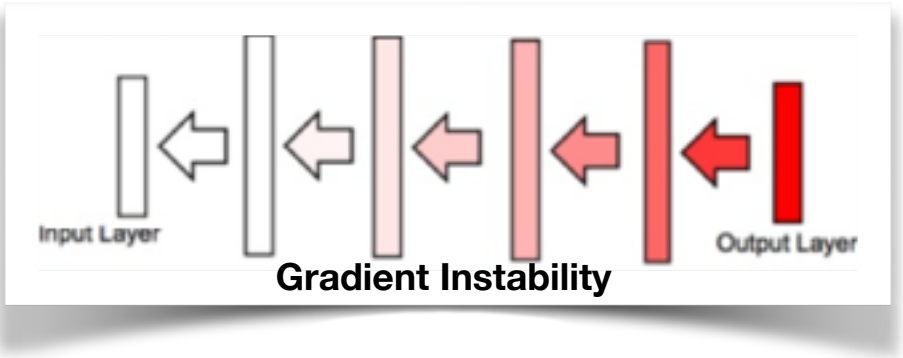
Read this: http://www.andreykurenkov.com/writing/a-brief-history-of-neural-nets-and-deep-learning/

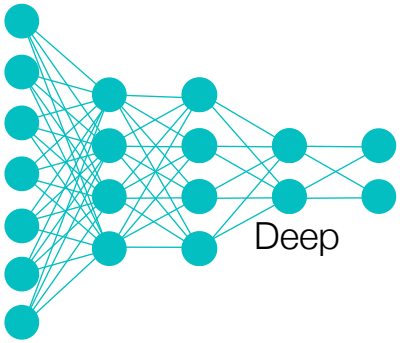
# History of Deep Learning: Winter


- AI Winter is coming:




Easy to train, performs on par with other methods




Hard to train, performs worse than other methods
**~chance (untrainable)**

1990's: Many companies founded on the promise of advanced ML and the internet…
2000's: Researcher have difficulty reconciling expressiveness with performance, cash in the market drops out, **no theory and no money**

Read this: http://www.andreykurenkov.com/writing/a-brief-history-of-neural-nets-and-deep-learning/

# Machine Learning Timeline (Neural Nets)

- 2004: Hinton secures funding from CIFAR based on his reputation
  - *eventually*: Canada would be savior for neural networks
  - Hinton rebrands: **Deep Learning**
- 2006: Hinton publishes paper on using pre-training and Restricted Boltzmann Machines
- 2007: Another paper: Deep networks are more efficient when pre-trained
  - RBMs not really the important part

1949, Hebb's Law
Close neuron fire together

1960, Widrow & Hoff
Adaline Network

1986, Rumelhart & Hinton
Back-propagation

2003, Vapnik
Kernel SVMs

1940 | 1960 | 1980 | 2000 | 2020

Period of Discovery | First AI Winter | Golden Age of NN | 2nd AI Winter | Age of Deep Learning

1943, McCulloch & Pitts
Logic Gates of The Mind

1957, Rosenblatt
Perceptron

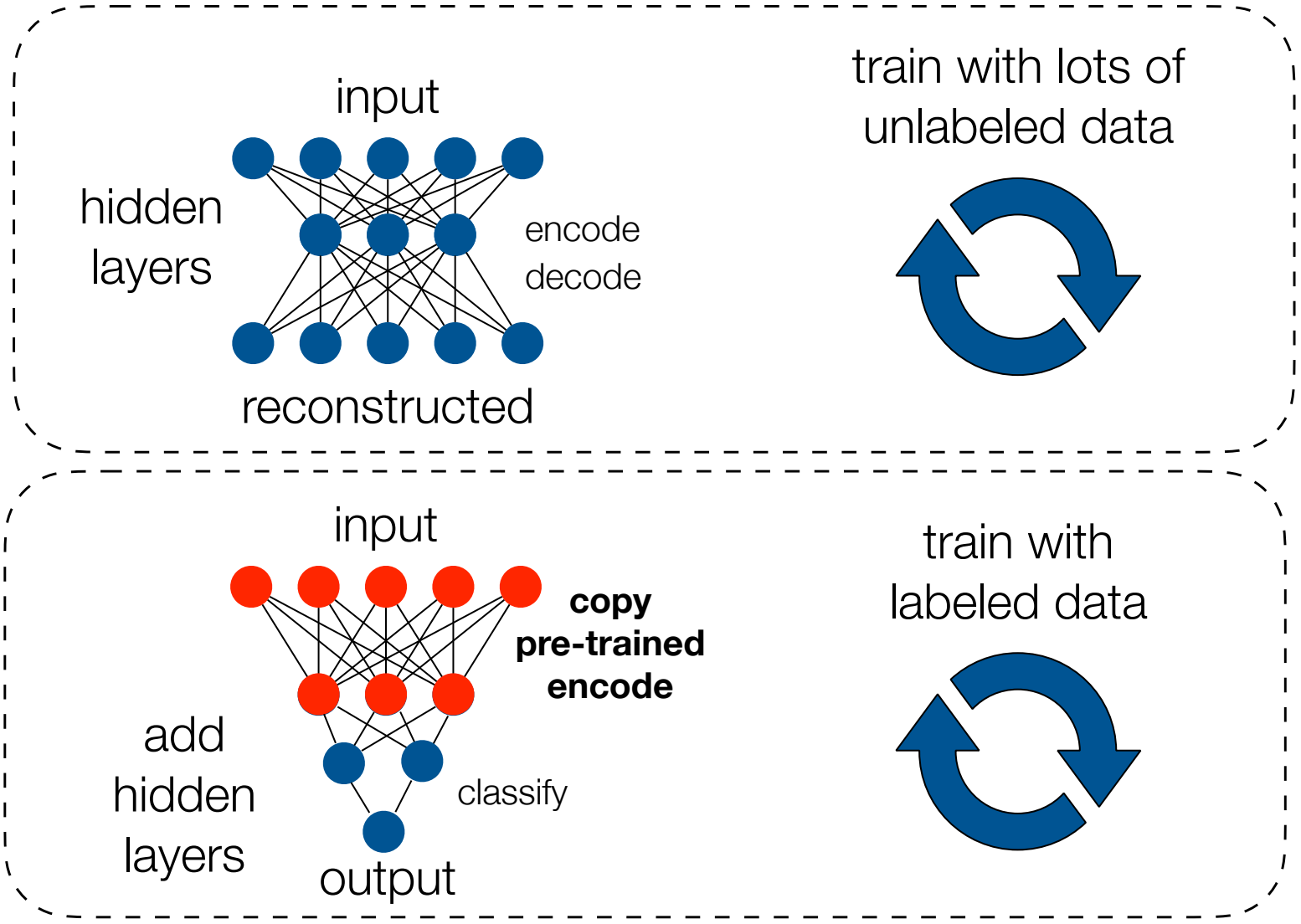
1969, Minsky & Papert
Linear Models are Doomed

2001, Breiman
Random Forests

Read this: http://www.andreykurenkov.com/writing/a-brief-history-of-neural-nets-and-deep-learning/

# Pre-training: still in the long winter

- auto-encoding (a form of pre-training)



# Still in the Long Winter

- 2009: Hinton's lab starts using GPUs, Also Andrew Ng
  - GPUs decrease training time by 70 fold…
- 2010: Hinton's and Ng's students go to internships with Microsoft, Google, IBM, and Facebook



Navdeep Jaitly
Google Brain Team


George Dahl
Google Brain Team




1949, Hebb's Law
Close neuron fire together

1960, Widrow & Hoff
Adaline Network

1986, Rumelhart & Hinton
Back-propagation

2003, Vapnik
Kernel SVMs

- Xbox Voice
- Android Speech Recognition
- IBM Watson
- DeepFace
- All of Baidu

1940 — 1960 — 1980 — 2000

Period of Discovery | First AI Winter | Golden Age of NN | 2nd AI Winter

1943, McCulloch & Pitts
Logic Gates of The Mind

1957, Rosenblatt
Perceptron

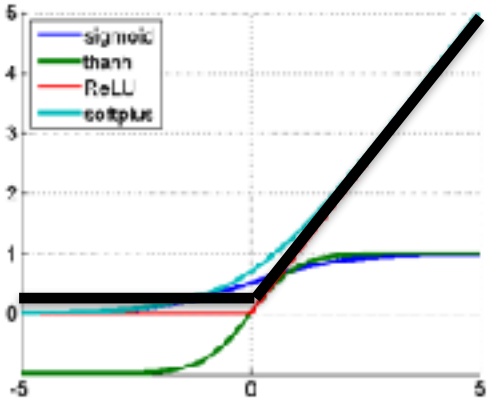
1969, Minsky & Papert
Linear Models are Doomed

2001, Breiman
Random Forests

Read this: http://www.andreykurenkov.com/writing/a-brief-history-of-neural-nets-and-deep-learning/

# Getting out of the long Winter

- 2011: Glorot and Bengio investigate more systematic methods for why past deep architectures did not work
  - **discover some interesting, simple fixes**: the type of neurons chosen and the selection of initial weights
  - do not require pre-training to get deep networks properly trained, just sparser representations and less complicated derivatives




ReLU: $f(x) = \max(0,x)$
$f'(x) = 1$ if $x>0$ else 0

1949, Hebb's Law
Close neuron fire together


1960, Widrow & Hoff
Adaline Network


1986, Rumelhart & Hinton
Back-propagation

2003, Vapnik
Kernel SVMs

1940 — 1960 — 1980 — 2000 — 2020

Period of Discovery | First AI Winter | Golden Age of NN | 2nd AI Winter | Age of Deep Learning

1943, McCulloch & Pitts
Logic Gates of The Mind

1957, Rosenblatt
Perceptron

1969, Minsky & Papert
Linear Models are Doomed

2001, Breiman
Random Forests

2011, Bengio
Init and ReLU

Read this: http://www.andreykurenkov.com/writing/a-brief-history-of-neural-nets-and-deep-learning/

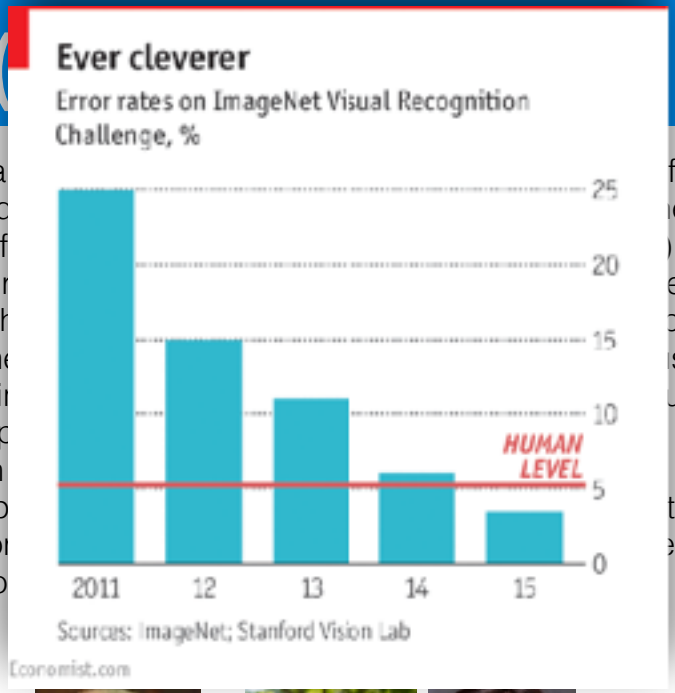
# Machine Learning Timeline (

- **ImageNet competition occurs**
- **Second place**: 26.2% error rate
- **First place**:
  - From Hinton's lab, uses convolutional network with ReLU and dropout
  - 15.2% error rate
- Computer vision adopts deep learning with convolutional neural networks en mass


"I ha
hanc
last f
so I
as sh
come
pacir
happ
from
skep
a cor
Visio

first
e
),
ef
on
s
ust

t as
er

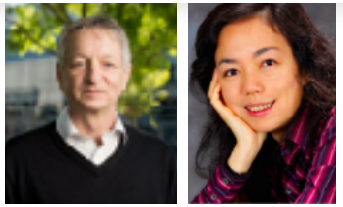



1949, Hebb's Law
Close neuron fire together

1960, Widrow & Hoff
Adaline Network

1986, Rumelhart & Hinton
Back-propagation

2003, Vapnik
Kernel SVMs



2012, Hinton, Fei-Fei Li
CNNs win ImageNet

1940 — 1960 — 1980 — 2000 — 2020

Period of Discovery | First AI Winter | Golden Age of NN | 2nd AI Winter | Age of Deep Learning

1943, McCulloch & Pitts
Logic Gates of The Mind

1957, Rosenblatt
Perceptron

1969, Minsky & Papert
Linear Models are Doomed

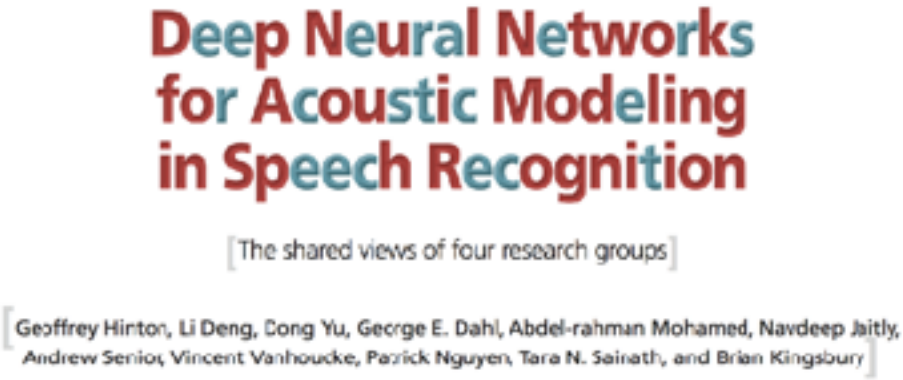
2001, Breiman
Random Forests

2011, Bengio
Init and ReLU

Read this: http://www.andreykurenkov.com/writing/a-brief-history-of-neural-nets-and-deep-learning/

# Machine Learning Timeline (Acoustics)

- 2012: Hinton Lab, Google, IBM, and Microsoft jointly publish paper, popularity for deep learning methods increases



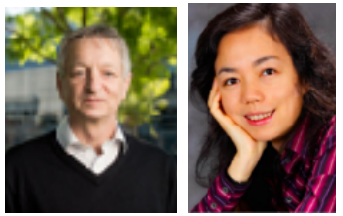

https://www.cs.toronto.edu/~gdahl/papers/deepSpeechReviewSPM2012.pdf

1949, Hebb's Law
Close neuron fire together

1960, Widrow & Hoff
Adaline Network

1986, Rumelhart & Hinton
Back-propagation

2003, Vapnik
Kernel SVMs



2012, Hinton, Fei-Fei Li
CNNs win ImageNet

1940 — 1960 — 1980 — 2000 — 2020

Period of Discovery | First AI Winter | Golden Age of NN | 2nd AI Winter | Age of Deep Learning

1943, McCulloch & Pitts
Logic Gates of The Mind

1957, Rosenblatt
Perceptron

1969, Minsky & Papert
Linear Models are Doomed

2001, Breiman
Random Forests

2011, Bengio
Init and ReLU

Read this: http://www.andreykurenkov.com/writing/a-brief-history-of-neural-nets-and-deep-learning/

# Machine Learning Timeline (Neural Nets)

- 2013: Andrew Ng and Google (BrainTeam)
  - run unsupervised feature creation on YouTube videos (becomes computer vision benchmark)

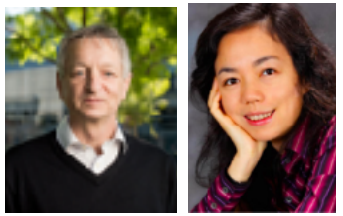
*The work resulted in unsupervised neural net learning of an unprecedented scale - 16,000 CPU cores powering the learning of a whopping 1 billion weights. The neural net was trained on Youtube videos, entirely without labels, and learned to recognize the most common objects in those videos.*

1949, Hebb's Law
Close neuron fire together

1960, Widrow & Hoff
Adaline Network

1986, Rumelhart & Hinton
Back-propagation

2003, Vapnik
Kernel SVMs



2012, Hinton, Fei-Fei Li
CNNs win ImageNet

1940 | 1960 | 1980 | 2000 | 2020

Period of Discovery | First AI Winter | Golden Age of NN | 2nd AI Winter | Age of Deep Learning

1943, McCulloch & Pitts
Logic Gates of The Mind

1957, Rosenblatt
Perceptron

1969, Minsky & Papert
Linear Models are Doomed

2001, Breiman
Random Forests

2011, Bengio
Init and ReLU

Read this: http://www.andreykurenkov.com/writing/a-brief-history-of-neural-nets-and-deep-learning/

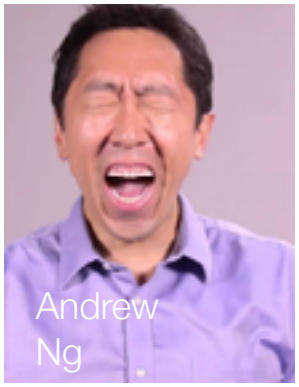
# A summary of the Deep Learning people:


Stayed at Univ. Montreal
Advises IBM


Heads Facebook AI Team


Univ. Toronto
Google


Stanford (HAI)
Former Chief Scien., AI/MLGoogle Cloud




Coursera
Baidu
Google


Stanford
Founded Coursera
MacArthur Genius


**Made Deep Learning Instruction Accessible**

deep learning

doi:10.1038/nature14539

ning

Geoffrey Hinton[4,5]

Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction. These methods have dramatically improved the state-of-the-art in speech recognition, visual object recognition, object detection and many other domains such as drug discovery and genomics. Deep learning discovers intricate structure in large data sets by using the backpropagation algorithm to indicate how a machine should change its internal parameters that are used to compute the representation in each layer from the representation in the previous layer. Deep convolutional nets have brought about breakthroughs in processing images, video, speech and

# Credit for Deep Learning

**Official ACM** @TheOfficialACM

Yoshua Bengio, Geoffrey Hinton and Yann LeCun, the fathers of #DeepLearning, receive the 2018 #ACMTuringAward for conceptual and engineering breakthroughs that have made deep neural networks a critical component of computing today. bit.ly/2HVJtdV



Machine learning is the science of credit assignment. The machine learning community itself profits from proper credit assignment to its members. The inventor of an important method should get credit for inventing it. She may not always be the one who popularizes it. Then the popularizer should get credit for popularizing it (but not for inventing it). Relatively young research areas such

# Review of Deep Learning History

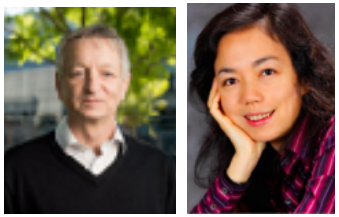
- Up to this point: back propagation saved AI winter for NN (Hinton and others!)
- 80's, 90's, 2000's: convolutional networks for image processing start to get deeper
  - but back propagation no longer does great job at training them
- SVMs and Random Forests gain traction…
  - The second AI winter begins, 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)

1949, Hebb's Law
Close neuron fire together

1960, Widrow & Hoff
Adaline Network

1986, Rumelhart & Hinton
Back-propagation

2003, Vapnik
Kernel SVMs



2012, Hinton, Fei-Fei Li
CNNs win ImageNet

1940 | 1960 | 1980 | 2000 | 2020

Period of Discovery | First AI Winter | Golden Age of NN | 2nd AI Winter | Age of Deep Learning

1943, McCulloch & Pitts
Logic Gates of The Mind

1957, Rosenblatt
Perceptron

1969, Minsky & Papert
Linear Models are Doomed

2001, Breiman
Random Forests

2011, Bengio
Init and ReLU

2015, Google
Tensorflow Open Source

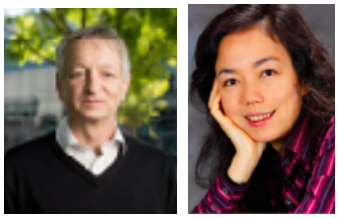
# The New Paradigm

1949, Hebb's Law
Close neuron fire together

1960, Widrow & Hoff
Adaline Network

1986, Rumelhart & Hinton
Back-propagation

2003, Vapnik
Kernel SVMs



2012, Hinton, Fei-Fei Li
CNNs win ImageNet

1940 | 1960 | 1980 | 2000 | 2020

Period of Discovery | First AI Winter | Golden Age of NN | 2nd AI Winter | Age of Deep Learning

1943, McCulloch & Pitts
Logic Gates of The Mind

1957, Rosenblatt
Perceptron

1969, Minsky & Papert
Linear Models are Doomed

2001, Breiman
Random Forests

2011, Bengio
Init and ReLU

2015, Google
Tensorflow Open Source


2015, Bengio, Schmidhuber
More Recurrent Nets

2018, Radford
GPT-1

2018 - 2022
Various GPT Papers

3rd AI Winter

future

2018 | 2020 | 2022 | 2024

Age of Deep Learning | Age of Large Language Models

2017, Vaswani
Attention is All You Need

2020, Brown
Few Shot Language Models


2022, OpenAI
ChatGPT

Hyped up Slop about promise of capability…

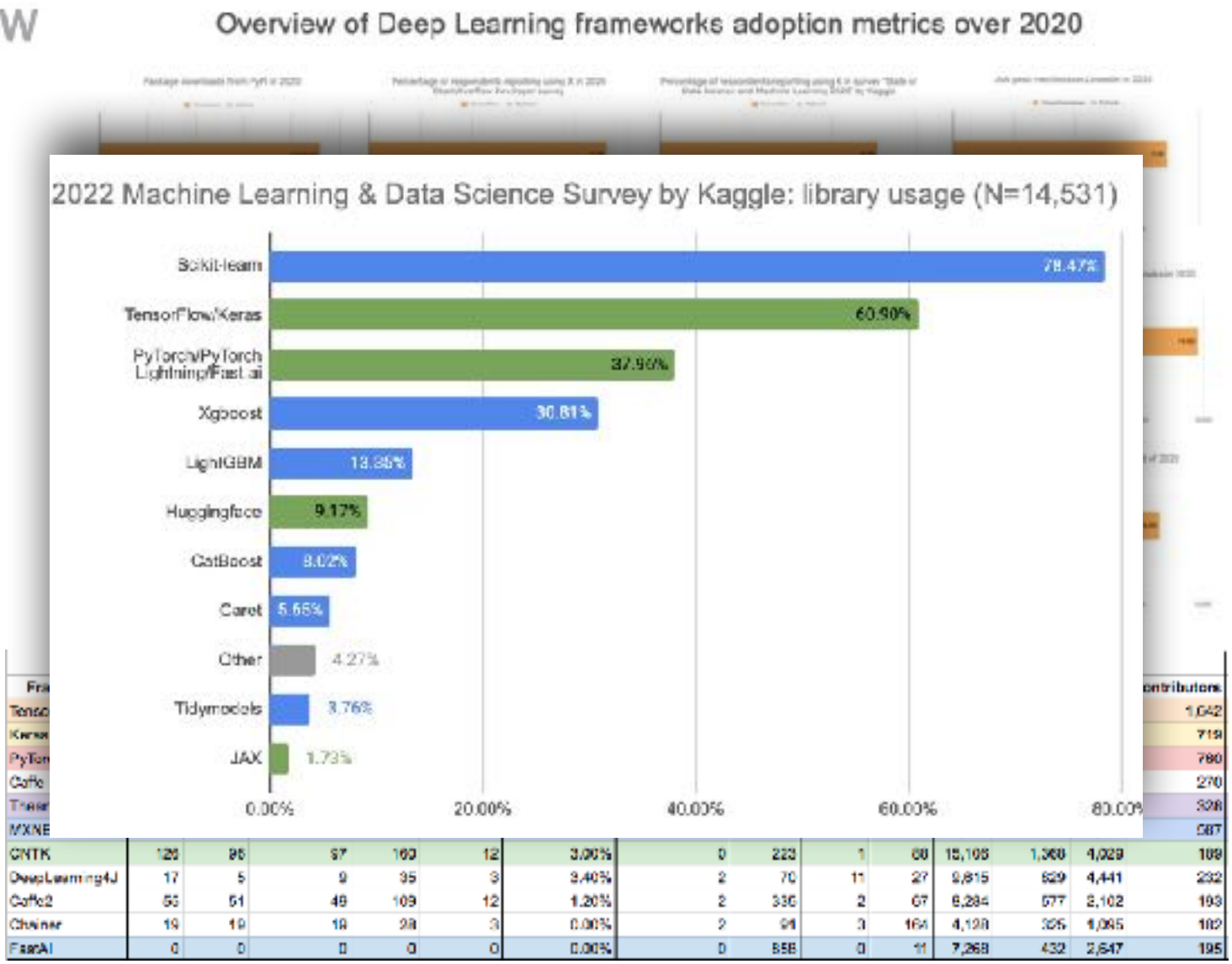
# Tensor Packages for Deep Learning

"Further discussion of it merely incumbers the literature and befogs the mind of fellow students."

- 2007: NIPS program committee rejects a paper on deep learning by *al. et.* Hinton because they already accepted a paper on deep learning and two papers on the same topic would be excessive.
- ~2009: A reviewer tells Yoshua Bengio that papers about neural nets have no place in ICML.
- ~2010: A CVPR reviewer rejects Yann LeCun's paper even though it beats the state-of-the-art. The reviewer says that it tells us nothing about computer vision because everything is learned.

# Options for Deep Learning Toolkits

1. TensorFlow
2. Keras
3. PyTorch
4. Caffe
5. theano
6. mxnet
7. CNTK
8. DL4J
9. Caffe2
10. Chainer
11. fast.ai




| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CNTK | 126 | 95 | 97 | 169 | 12 | 3.00% | 0 | 223 | 1 | 69 | 15,105 | 1,360 | 4,029 | 109 |
| DeepLearning4J | 17 | 5 | 9 | 35 | 3 | 9.40% | 2 | 70 | 11 | 27 | 2,815 | 529 | 4,441 | 232 |
| Caffe2 | 55 | 51 | 49 | 109 | 12 | 1.20% | 2 | 330 | 2 | 67 | 6,284 | 577 | 2,102 | 193 |
| Chainer | 19 | 19 | 19 | 28 | 3 | 0.00% | 2 | 91 | 3 | 164 | 4,139 | 325 | 1,085 | 102 |
| FastAI | 0 | 0 | 0 | 0 | 0 | 0.00% | 0 | 558 | 0 | 11 | 7,268 | 432 | 2,647 | 195 |

# Tensorflow

- Open sourced library from Google
- Second generation release from Google Brain
  - supported for Linux, Unix, Windows
  - Also works on Android/iOS
- Released November 9th, 2015
  - (this class first offered January 2016)
- PyTorch released January 2017

# Programmatic creation

- Most toolkits use python to build a **computation graph** of operations
  - Build up computations
  - Execute computations
- **Most Toolkits Support:**
  - tensor creation
  - functions on tensors
  - automatic differentiation
- Tensors are just multidimensional arrays
  - like in Numpy
    - scalars (biases and constants)
    - vectors (e.g., input arrays)
    - 2D matrices (e.g., images)
    - 3D matrices (e.g., color images)
    - 4D matrices (e.g., batches of color images)

# Tensor basic functions

```
a = tf.constant(5.0)

b = tf.constant(6.0)

c = a * b
```

- Easy to define operations on tensors

| Numpy | TensorFlow |
|---|---|
| `a = np.zeros((2,2)); b = np.ones((2,2))` | `a = tf.zeros((2,2)), b = tf.ones((2,2))` |
| `np.sum(b, axis=1)` | `tf.reduce_sum(a,reduction_indices=[1])` |
| `a.shape` | `a.get_shape()` |
| `np.reshape(a, (1,4))` | `tf.reshape(a, (1,4))` |
| `b * 5 + 1` | `b * 5 + 1` |
| `np.dot(a,b)` | `tf.matmul(a, b)` |
| `a[0,0], a[:,0], a[0,:]` | `a[0,0], a[:,0], a[0,:]` |

Also supports convolution: `tf.nn.conv2d, tf.nn.conv3D`

https://cs224d.stanford.edu/lectures/CS224d-Lecture7.pdf

# Tensor neural network functions

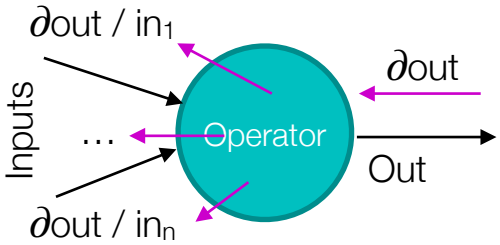
- Easy to define operations on layers of networks

```
• relu(features, name=None)
• bias_add(value, bias, data_format=None, name=None)
• sigmoid(x, name=None)
• tanh(x, name=None)
• conv2d(input, filter, strides, padding)
• conv1d(value, filters, stride, padding)
• conv3d(input, filter, strides, padding)
• conv3d_transpose(value, filter, output_shape, strides)
• sigmoid_cross_entropy_with_logits(logits, targets)
• softmax(logits, dim=-1)
• log_softmax(logits, dim=-1)
• softmax_cross_entropy_with_logits(logits, labels, dim=-1)
```

- Each function created *knows its gradient*
- **Automatic Differentiation** is just **chain rule**
- But… lets start simple…

# Auto Differentiation (under the hood)

```
class add(Operator):
    """Binary addition operation."""
    def __init__(self, a, b):
        super().__init__()
        self.inputs=[a, b]

    def forward(self, a, b):
        return a+b

    def backward(self, a, b, dout):
        return dout, dout
```




Backward: derivative for each input

```
class matmul(Operator):
    """Binary multiplication"""
    def __init__(self, a, b):
        super().__init__()
        self.inputs=[a, b]

    def forward(self, a, b):
        return a@b

    def backward(self, a, b, dout):
        return dout@b.T, a.T@dout
```

```
def forward_pass(order, feed_dict={}):
    """ Performs the forward pass, returning output of graph.
    Args:
        order: a topologically sorted array of nodes
        feed_dict: a dictionary values for placeholders.
    Returns:
        1. the final result of the forward pass.
        2. edits the graph to fill in its values.
    """
    for node in order:
        tmp = [prev_node.value for prev_node in node.inputs]
        node.value = node.forward(*tmp)

    ret
```

**Could also have GPU implementations**

```
def ba
    """ Perform the backward pass to retrieve gradients.
    gradients of nodes as listed in same order as input """
    vis = set()
    order[-1].gradient = 1
    for node in reversed(order):
        inputs = node.inputs
        grads = node.backward(*[x.value for x in inputs],
                              dout=node.gradient)
        for inp, grad in zip(inputs, grads):
            if inp not in vis:
              inp.gradient = grad
            else:
              inp.gradient += grad
            vis.add(inp)
    return [node.gradient for node in order]
```

https://towardsdatascience.com/build-your-own-automatic-differentiation-program-6ecd585eec2a/

# Tensor function evaluation

```
import tensorflow as tf

a = tf.constant(5.0)
b = tf.constant(6.0)

c = a*b

// pre tensorflow 2
with tf.Session() as sess:
    print(sess.run(c))
    print(c.eval())

// post tensorflow 2
print(c)
```

output = 30

- Easy to define operations on tensors
  - constant
  - variables
  - placeholders
- Nothing evaluated until the output is needed
- Tensorflow now run in default eager execution

https://cs224d.stanford.edu/lectures/CS224d-Lecture7.pdf

# Computation Graph with Code

```
import tensorflow as tf
X = tf.Placeholder()
y = tf.Placeholder()
```

$$J(\mathbf{W}) = \frac{1}{N}\sum_{i}^{N}(y^{(i)} - \underbrace{(\mathbf{W}\cdot\mathbf{x}^{(i)} + \mathbf{b})}_{y_{pred}})^2$$

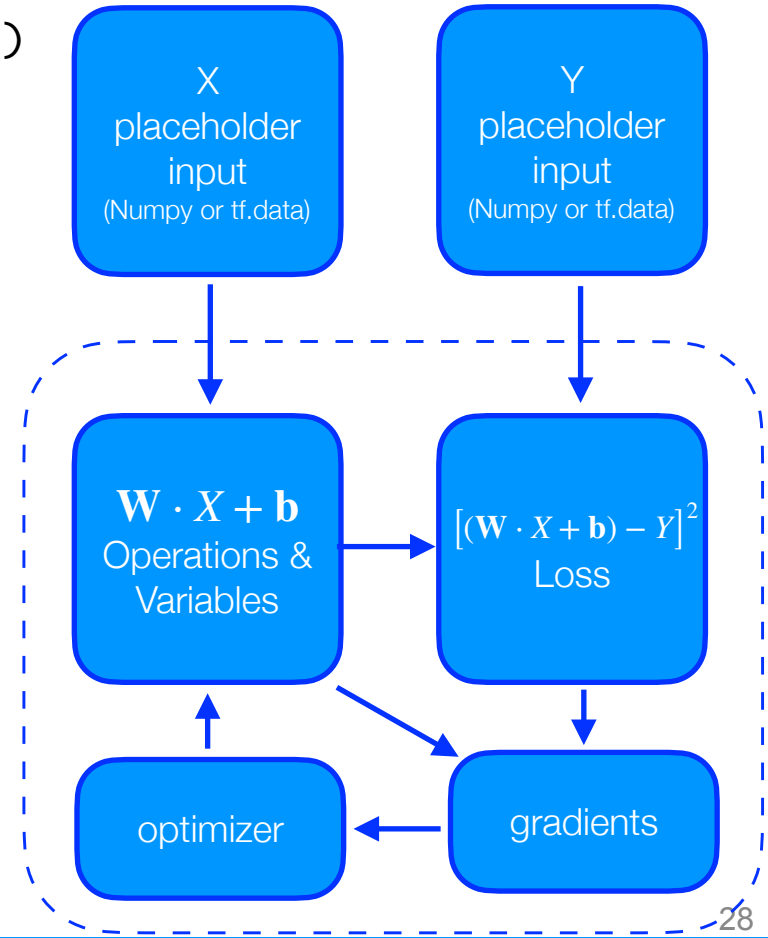
1. **Setup** Variables and computations

```
W = tf.Variable("weights", (1,num_features),
        initializer=tf.random_normal_initializer())
b = tf.Variable("bias", (1,),
        initializer=tf.constant_initializer(0.0))

def feedforward(X,y):
    y_pred = tf.matmul(X,W) + b
    loss = tf.reduce_sum((y-y_pred)**2)
    return loss
```

2. Add **optimization** operation to computation graph
   Adjusts variables (W, b) to minimize loss with auto differentiation

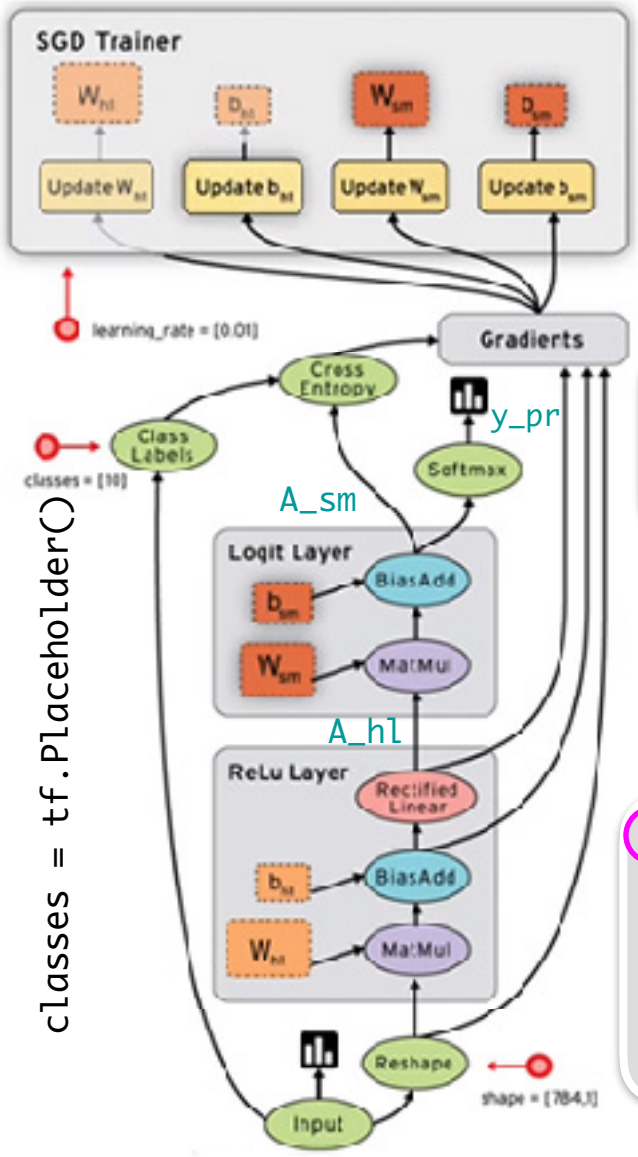
```
opt = tf.train.AdamOptimizer()

tf.initialize_all_variables()
feedforward(X_numpy, y_numpy)
    … track gradients and variables …
opt.apply_gradients(zip(grads, train_vars))
```

3. **Run graph operation** once, → one optimization update on all variables




http://www.datasciencecentral.com/profiles/blogs/google-open-source-tensorflow

# Computation Graph, Two Layer Network




```
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))
```

**tf.data**

http://www.datasciencecentral.com/profiles/blogs/google-open-source-tensorflow

# Tensorflow Simplification

- **Self Test**: Can the syntax be simplified?
  - (A) **Yes**, we could write a generic mini-batch optimization computation graph, then use it for arbitrary graph instructions
  - (B) **Yes**, but we need to learn the Keras API, which can be mixed with tensorflow operations
  - (C) **Yes**, but we need to understand how to access the gradients to apply them, a lot like PyTorch
  - (D) **All of the above**

# End of Session

- Next Time:
  - Introduction to TensorFlow
  - Wide and Deep Networks