Neural Networks 101: Implementing Feedforward Neural Nets using TensorFlow

Lu Lu

Aug 3, 2018 @Crunch Seminar

Feedforward Neural Nets (FNN)

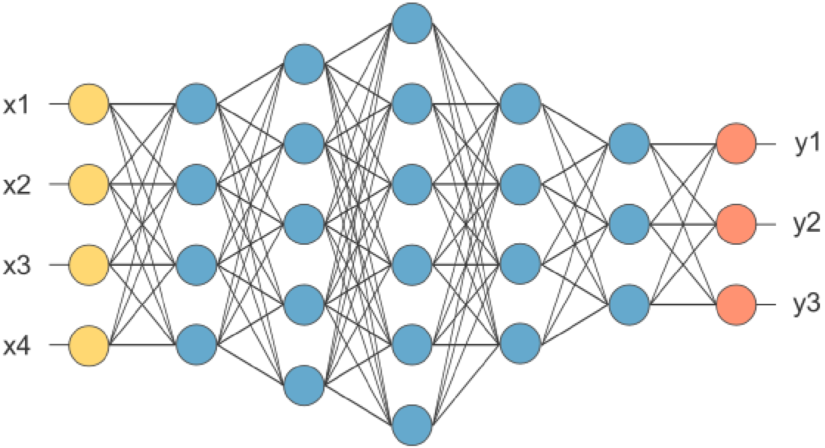


Figure 1

Feedforward Neural Nets (FNN)

$\mathcal{N} : \mathbb{R}^{d_{in}} \to \mathbb{R}^{d_{out}}$ with L layers and N^l neurons in the layer l

$$\mathbf{x}^l = \sigma(\mathbf{h}^l) \qquad \mathbf{h}^l = \mathbf{W}^l \mathbf{x}^{l-1} + \mathbf{b}^l \quad \text{for } l = 1, \dots, L-1$$

$$\mathbf{y} \equiv \mathbf{x}^L = \mathbf{h}^L = \mathbf{W}^L \mathbf{x}^{L-1} + \mathbf{b}^L$$

- $\mathbf{x}^0 \in \mathbb{R}^{d_{in}}$: input
- $\mathbf{W}^l$: weight matrix ($N^l \times N^{l-1}$) in the layer l
- $\mathbf{b}^l \in \mathbb{R}^{N^l}$: biases in the layer l
- σ: $\mathbb{R} \to \mathbb{R}$, component-wise activation function (nonlinear)
- $\mathbf{x}^l \in \mathbb{R}^{N^l}$: neural activity in the layer l

$$\mathbf{y} \equiv \mathbf{x}^L = \mathbf{W}^L \sigma(\mathbf{W}^{L-1} \sigma(\dots \sigma(\mathbf{W}^2 \sigma(\mathbf{W}^1 \mathbf{x}^0 + \mathbf{b}^1) + \mathbf{b}^2)) + \mathbf{b}^{L-1}) + \mathbf{b}^L$$

What activations CAN we use IN THEORY?[2]

- $\sigma : \mathbb{R} \to \mathbb{R}$ is a generalized sigmoidal function, if

$$\sigma(x) \to \begin{cases} 1 & \text{as } x \to +\infty \\ 0 & \text{as } x \to -\infty \end{cases}.$$

- σ is a Tauber-Wiener (TW) function, if all the linear combinations $\sum_{i=1}^{N} c_i \sigma(\lambda_i x + \theta_i)$, $i = 1, \ldots, N$, are dense in every $C[a, b]$.
- If σ is a **bounded** generalized sigmoidal function, then $\sigma \in (\mathrm{TW})$
- Suppose that $\sigma \in C(\mathbb{R}) \cap S'(\mathbb{R})$[1], then $\sigma \in (\mathrm{TW})$ iff σ is not a polynomial
- If $\sigma \in (\mathrm{TW})$, then every measurable function f can be approximated arbitrarily well by a single-hidden-layer FNN.
- **Any continuous, slowly increasing, non-polynomial function can be used as an activation.**

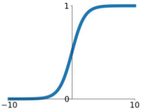
[1] Contains all slowly increasing functions (polynomially growing)

[2] Chen & Chen, IEEE Trans Neural Netw, 1993; Chen & Chen, IEEE Trans Neural Netw, 1995.

What activations SHOULD we use IN REALITY?[3]

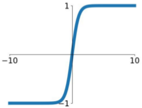
Sigmoid

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



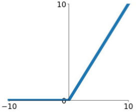
tanh

$\tanh(x)$



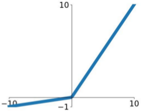
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

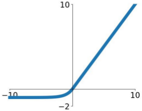


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Suggestions:

- Never use Sigmoid
- For very deep NN (>10 layers), try ReLU-based activations first

[3] LeCun et al., Neural Netw, 1998; Glorot & Bengio, AISTATS, 2010; Glorot et al., AISTATS, 2011.

Machine learning frameworks

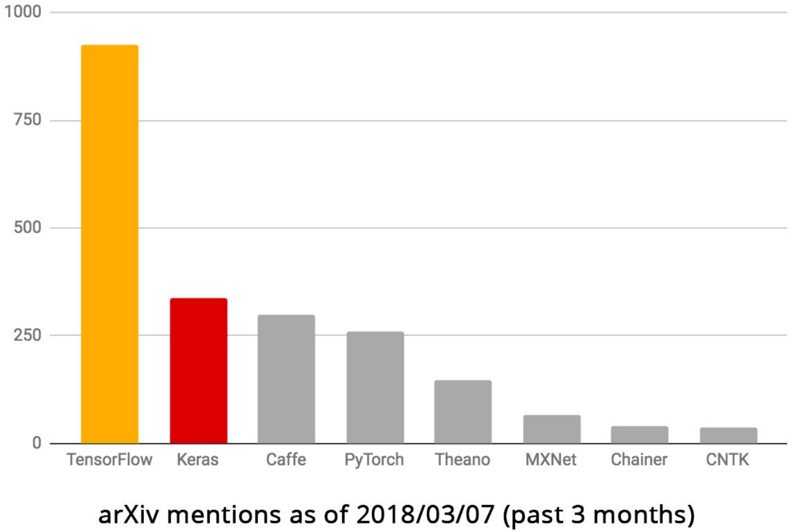


Figure 2

What is TensorFlow?

- Open sourced by Google in November 2015
- Library for **numerical computation**
- **NOT** provide out-of-the-box machine learning solutions
- Tensor: ~~geometric objects that describe linear relations between geometric vectors, scalars, and other tensors~~ n-dimensional matrix

TensorFlow

```
import tensorflow as tf
```

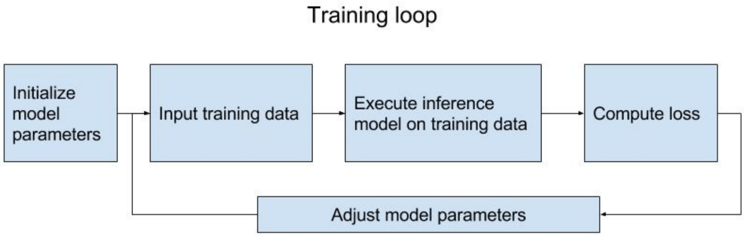
- data type: `tf.float32`, `tf.float64`
- Inputs: `tf.placeholder`
- Variables to be optimized: `tf.Variable`
- Math operations
  - Multiplies matrix a by matrix b: `tf.matmul(a, b)`
  - `tf.nn.tanh`, `tf.nn.relu`
  - mean: `tf.reduce_mean`
- Session

```
# Build a NN
...
# Launch, init, run
sess = tf.Session()
sess.run(tf.global_variables_initializer())
sess.run(..., feed_dict={...})
```

Hands-on: Build your own FNN

Figure 3: Mission Impossible?

What is next?



After building a neural network, train, i.e., optimize **W** and **b** using data.

- Define a loss to be minimized
- Initialize the net
- Choose an optimizer

Loss

Traing data set: $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$

- Mean absolute error (MAE) or L1

$$\frac{1}{n}\sum_{i=1}^{n} |\mathcal{N}(x_i) - y_i|$$

- Mean squared error (MSE) or L2

$$\frac{1}{n}\sum_{i=1}^{n} (\mathcal{N}(x_i) - y_i)^2$$

```
tf.reduce_mean((y_pred - y_true)**2)
```

- ...

How to initialize the weights?

Weights are randomly sampled (zero mean). $\mathbf{b} = 0$.

Initializer	Var[w]	Activation
Glorot uniform/normal[4]	$2/(fan_{in} + fan_{out})$	tanh
He normal[5]	$2/fan_{in}$	ReLU
LeCun normal[6]	$1/fan_{in}$	SeLU
Orthogonal[7]	-	all
LSUV[8]	-	all

- fan_{in}: the number of input units of the layer
- fan_{out}: the number of output units of the layer

[4] Glorot & Bengio, AISTATS, 2010.
[5] He et al., ICCV, 2015.
[6] LeCun et al., Neural Netw, 1998; Klambauer et al., NIPS, 2017.
[7] Saxe et al., ICLR, 2014.
[8] Mishkin & Matas, ICLR, 2015.

How to optimize?

Optimizers

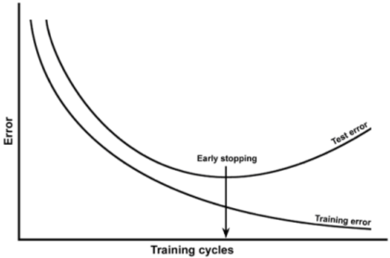
- SGD: $w_{t+1} = w_t - \eta \nabla_w loss(w)$
- SGDNesterov[9]: momentum
- AdaGrad[10]: adaptive per-parameter learning rates
- AdaDelta[11], RMSProp[12]: extensions of AdaGrad
- Adam[13]: adaptive & momentum
- ...

```
learning_rate = ...
loss = ...
opt = tf.train.AdamOptimizer(learning_rate)
train = opt.minimize(loss)
```

---

[9] Sutskever et al., ICML, 2013.
[10] Duchi et al., JMLR, 2011.
[11] Zeiler, arXiv, 2012.
[12] Hinton, csc321, 2014.
[13] Kingma & Ba, ICLR, 2015.

What else?



- Overfitting
  - Early stopping: Beautiful FREE LUNCH[14]
  - L1/L2 regularization: $\lambda \sum_w |w|^2$
  - Dropout[15]
- Normalization
- ...

[14] Hinton, NIPS, 2015.
[15] Srivastava et al., JMLR, 2014.

Hands-on: Training & Predicting

Figure 4: You can predict the unknown in a snap.