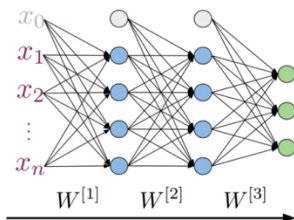


Neural Network for Sentiment Analysis

Neural Networks for Sentiment Analysis

Forward propagation



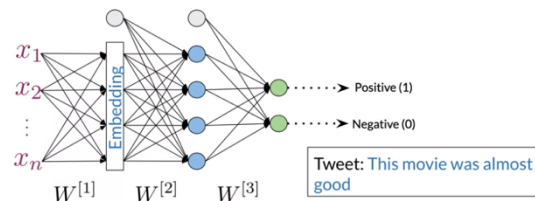
$a^{[i]}$ Activations ith layer

$$a^{[0]} = X$$

$$z^{[i]} = W^{[i]} a^{[i-1]}$$

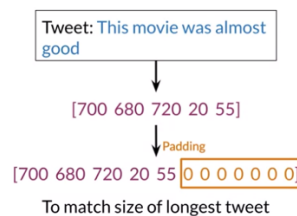
$$a^{[i]} = g^{[i]}(z^{[i]})$$

Neural Networks for sentiment analysis



Initial Representation

Word	Number
a	1
able	2
about	3
...	...
hand	615
...	...
happy	621
...	...
zebra	1000



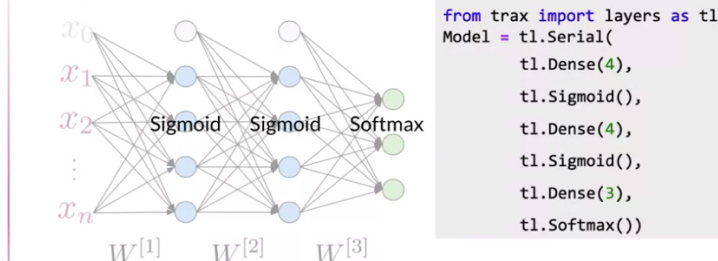
Trax: Neural Networks

Trax is built on TensorFlow

Advantages:

- Efficient computation
- Parallel computing by running gear models in multiple m/c
- Keeps a record of all algebraic operations on your NN in order of computation.
- Able to compute the gradients of the model automatically.

Neural Networks in Trax



Why TRAX

TensorFlow → Translate → **Tensor2Tensor** → Trax What do you want from a deep learning library'?

Accelerating Deep Learning Research with the Tensor2Tensor Library

Monday, June 19, 2017

Posted by Łukasz Kaiser, Senior Research Scientist, Google Brain Team

Translation Model	Training time	BLEU (difference from baseline)
Transformer (T2T)	3 days on 8 GPUs	28.4 (+7.8)
SliceNet (T2T)	6 days on 32 GPUs	26.1 (+5.5)
GNMT + Mixture of Experts	1 day on 64 GPUs	26.0 (+5.4)
ConvS2S	18 days on 1 GPU	25.1 (+4.5)
GNMT	1 day on 96 GPUs	24.6 (+4.0)
ByteNet	8 days on 32 GPUs	23.8 (+3.2)
MOSES (phrase-based baseline)	N/A	20.6 (+0.0)

- Makes programmers efficient

- Runs code fast

GPU (8x V100)	on-demand: \$19.84 / hour	preemptible: \$5.92 / hour
TPU (8x v3)	on-demand: \$8 / hour	preemptible: \$1.40 / hour

Trax Adam

Trax makes programmers efficient

- Bottom-up clean re-design
- Easy to debug, you can read the code
- Full models with dataset bindings included
- Main models regression-tested daily

Is there a price to pay?

- Backwards compatibility

```
class Adam(Optimizer):
    """Adam optimizer."""

    def init(self, param):
        m = np.zeros_like(param)
        v = np.zeros_like(param)
        return m, v

    def update(self, step, grads, param, slots, opt_params):
        m, v = slots
        learning_rate, b1, b2, eps = opt_params
        m = (1 - b1) * grads + b1 * m # First moment estimate.
        v = (1 - b2) * (grads ** 2) + b2 * v # Second moment estimate.
        mhat = m / (1 - b1 ** (step + 1)) # Bias correction.
        vhat = v / (1 - b2 ** (step + 1)) # Bias correction.
        param = param - (
            learning_rate * mhat / (np.sqrt(vhat) + eps)).astype(param.dtype)
        return param, (m, v)
```

$$\begin{aligned}
 & m_0 \leftarrow 0 \text{ (Initialize 1st moment vector)} \\
 & v_0 \leftarrow 0 \text{ (Initialize 2nd moment vector)} \\
 & t \leftarrow 0 \text{ (Initialize timestep)} \\
 & \text{while } \theta_t \text{ not converged do} \\
 & \quad t \leftarrow t + 1 \\
 & \quad g_t \leftarrow \nabla_{\theta} f(\theta_{t-1}) \text{ (Get gradients w.r.t. stochastic objective at timestep } t) \\
 & \quad \hat{m}_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t \text{ (Update biased first moment estimate)} \\
 & \quad \hat{v}_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2 \text{ (Update biased second raw moment estimate)} \\
 & \quad \tilde{m}_t \leftarrow \hat{m}_t / (1 - \beta_1) \text{ (Compute bias-corrected first moment estimate)} \\
 & \quad \tilde{v}_t \leftarrow \hat{v}_t / (1 - \beta_2) \text{ (Compute bias-corrected second raw moment estimate)} \\
 & \quad \theta_t \leftarrow \theta_{t-1} - \alpha \cdot \tilde{m}_t / (\sqrt{\tilde{v}_t} + \epsilon) \text{ (Update parameters)} \\
 & \text{end while}
 \end{aligned}$$

Trax runs code fast

- Designed to use a JIT compiler with JAX and XLA
- JAX: fastest Transformer in MLPerf 2020 (JAX: 0.26, TF: 0.35, pyTorch: 0.62)
- No code changes at all between CPU, GPU and TPU, preemptible training
- Tested with TPUs on colab too:

Notebook settings

Hardware accelerator
TPU

GPU (8x V100)	on-demand: \$19.84 / hour	preemptible: \$5.92 / hour
TPU (8x v3)	on-demand: \$8 / hour	preemptible: \$1.40 / hour

Trax: Background

Why Trax and not TensorFlow or PyTorch?

TensorFlow and PyTorch are both extensive frameworks that can do almost anything in deep learning. They offer a lot of flexibility, but that often means verbosity of syntax and extra time to code.

Trax is much more concise. It runs on a TensorFlow backend but allows you to train models with 1 line commands. Trax also runs end to end, allowing you to get data, model and train all with a single terse statements. This means you can focus on learning, instead of spending hours on the idiosyncrasies of big framework implementation.

Why not Keras then?

Keras is now part of Tensorflow itself from 2.0 onwards. Also, trax is good for implementing new state of the art algorithms like Transformers, Reformers, BERT because it is actively maintained by Google Brain Team for

advanced deep learning tasks. It runs smoothly on CPUs, GPUs and TPUs as well with comparatively lesser modifications in code.

How to Code in Trax

Building models in Trax relies on 2 key concepts:- **layers** and **combinators**. Trax layers are simple objects that process data and perform computations. They can be chained together into composite layers using Trax combinators, allowing you to build layers and models of any complexity.

Trax, JAX, TensorFlow and Tensor2Tensor

You already know that Trax uses Tensorflow as a backend, but it also uses the JAX library to speed up computation too. You can view JAX as an enhanced and optimized version of numpy.

Watch out for assignments which import `import trax.fastmath.numpy as np`. If you see this line, remember that when calling `np` you are really calling Trax's version of numpy that is compatible with JAX.

As a result of this, where you used to encounter the type `numpy.ndarray` now you will find the type `jax.interpreters.xla.DeviceArray`.

Tensor2Tensor is another name you might have heard. It started as an end to end solution much like how Trax is designed, but it grew unwieldy and complicated. So you can view Trax as the new improved version that operates much faster and simpler.

Resources

- Trax source code can be found on Github: [Trax](#)
- JAX library: [JAX](#)
- Official maintained documentation - [trax-ml](#) not to be confused with this [TraX](#)

```
#!pip install trax==1.3.1
import numpy as np # regular ol' numpy
from trax import layers as tl # core building block
from trax import shapes # data signatures: dimensionality and type
from trax import fastmath # uses jax, offers numpy on steroids
!pip list | grep trax
'/opt/conda/bin/python -m pip install --upgrade pip'
```

Trax: Layers Classes and Subclasses

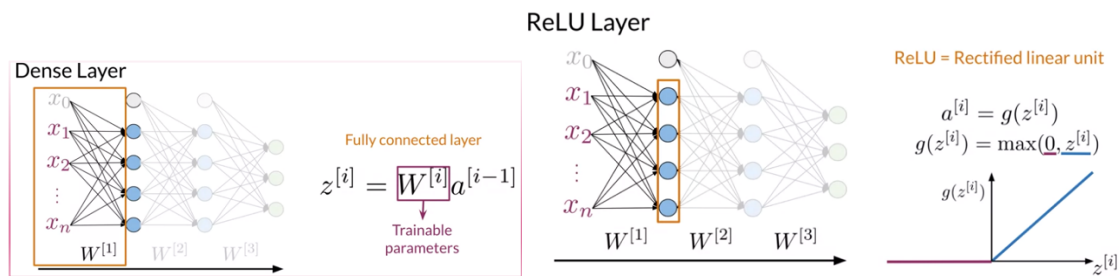
Subclasses

```
class MyClass(Object):
    def __init__(self,y):
        self.y = y
    def my_method(self,x):
        return x + self.y
    def __call__(self,x):
        return self.my_method(x)
```

```
class SubClass(MyClass):
    def my_method(self,x):
        return x + self.y**2

f = SubClass(7)
print(f(3))
52
```

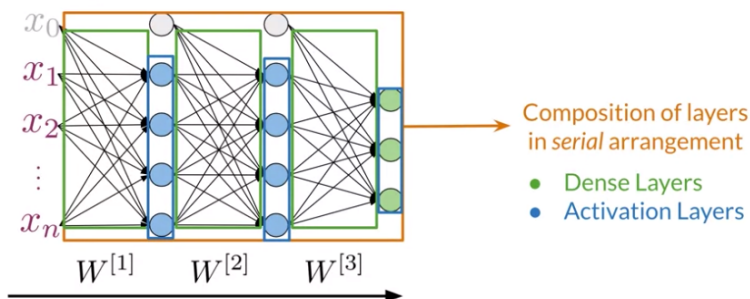
Dense and ReLU Layers



Serial Layer

A serial layer is a composition of sublayers that operates sequentially to perform the forward computation of your entire model.

Serial Layer



Other Layers

Embedding Layer – Embedding is trainable using an embedding layer. An embedding layer takes an index assigned to each word from your vocabulary and maps it to a representation of that word with a determined dimension. In this example, embedding of size equal to two.

Embedding Layer

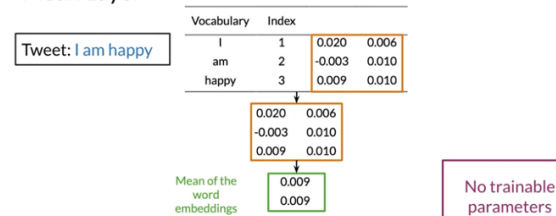
Vocabulary	Index		
I	1	0.020	0.006
am	2	-0.003	0.010
happy	3	0.009	0.010
because	4	-0.011	-0.018
learning	5	-0.040	-0.047
NLP	6	-0.009	0.050
sad	7	-0.044	0.001
not	8	0.011	-0.022

Trainable weights

Vocabulary x Embedding

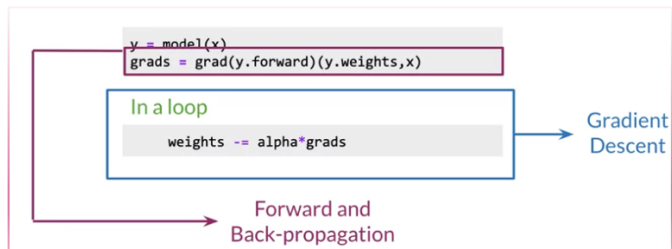
Mean Layer – Mean layer gives a vector representation.

Mean Layer



Training

- Computing gradients in trax
- Grad()
 - $F = 3x^2 + x$; `gradf = trax.math.grad(f)`
- `Y=model(x); grads = grad(y.forward)(y.weights, x); weights -=alpha*grads`



Trax: Reading

Official Trax documentation maintained by the Google Brain team:

<https://trax-ml.readthedocs.io/en/latest/>

Trax source code on GitHub:

<https://github.com/google/trax>

JAX library:

<https://jax.readthedocs.io/en/latest/index.html>
