**Notes of Study on Dec 14, 2022 (Wed)**

Based on lecture notes of Stanford

Module 1: Neural Networks

[Neural Networks Part 1: Setting up the Architecture](https://cs231n.github.io/neural-networks-1/)

model of a biological neuron, activation functions, neural net architecture, representational power

And

<https://www.youtube.com/watch?v=d14TUNcbn1k> (starts at 55 mins)

1. **A course computational model of neuron and its assumptions**
   1. Signals that travel along the axons (e.g. ***x0***) interact multiplicatively (e.g. ***w0x0***) with the dendrites of the other neuron based on the synaptic strength at that synapse (e.g. ***w0***).
   2. The idea is that **the synaptic strengths (the weights w) are learnable** and control the strength of influence (and its direction: excitory (positive weight) or inhibitory (negative weight)) of one neuron on another.
   3. Dendrites carry the signal to the cell body where they all get summed. **If the final sum is above a certain threshold, the neuron can fire, sending a spike along its axon.**
   4. We assume that the precise timings of the spikes do not matter, and that only the frequency of the firing communicates information. Based on this rate code interpretation, we **model the firing rate of the neuron with an activation function f**, which represents the frequency of the spikes along the axon.

ダイアグラム

自動的に生成された説明

* 1. Note the **limitation** of the course neuron model:
     1. In fact, there are many **different types of neurons, each with different properties**.
     2. The dendrites in biological neurons perform complex **nonlinear computations**.
     3. The synapses are not just a single weight, they’re a complex **non-linear dynamical system.**
     4. The exact timing of the output spikes in many systems is known to be important, suggesting that **the rate code approximation may not hold.**
  2. **A single neuron** can be used to implement **a binary classifier** (e.g. binary Softmax or binary SVM classifiers.
     1. A neuron has the capacity to “like” (activation near one) or “dislike” (activation near zero) certain linear regions of its input space.
     2. Hence, with an appropriate loss function on the neuron’s output, we can turn a single neuron into a linear classifier.
     3. R**egularization interpretation**: The regularization loss in both SVM/Softmax cases could in this biological view be interpreted as **gradual forgetting,** since it would have the effect of driving all synaptic weights w towards zero after every parameter update.

1. Activation functions
   1. **Sigmoid** and **Tanh: never use Sigmoid!** 
      1. **Sigmoid**

Output varies in (0, 1).

* + 1. **Tanh**

Output varies in (-1, 1), zero-centered.

* + 1. Sigmoid and Tanh saturate and kill gradients.
       1. **When the neuron’s activation saturates at either tail of 0 or 1, the gradient at these regions is almost zero.** Recall that during backpropagation, this (local) gradient will be multiplied to the gradient of this gate’s output for the whole objective. Therefore, if the local gradient is very small, it will effectively “kill” the gradient and almost no signal will flow through the neuron to its weights and recursively to its data.
       2. If the initial weights are too large then most neurons would become saturated and the network will barely learn.

グラフィカル ユーザー インターフェイス

低い精度で自動的に生成された説明

* 1. **ReLU: the most common activation function.**

ダイアグラム, 多角形

中程度の精度で自動的に生成された説明

* + 1. **Greatly accelerate the convergence of stochastic gradient descent** compared to the sigmoid/tanh functions. It is argued that this is due to its linear, non-saturating form.
    2. **Pay extra attention to the learning rate!**
       1. A large gradient flowing through a ReLU neuron could cause the weights to update in such a way that the neuron will never activate on any datapoint again. If this happens, then the gradient flowing through the unit will forever be zero from that point on.
       2. That is, the ReLU units can irreversibly die during training since they can get knocked off the data manifold.
       3. **You may find that as much as 40% of your network can be “dead” (i.e. neurons that never activate across the entire training dataset) if the learning rate is set too high**.
    3. Variation form of activation functions inspired by ReLU:
       1. **Leaky ReLU**

**where α is a small constant.**

* + - 1. **Maxout**

Note both ReLU and Leaky ReLU are a special case of this form (for example, for ReLU we have w1,b1=0).

And Maxout neuron therefore enjoys all the benefits of a ReLU unit (linear regime of operation, no saturation) and does not have its drawbacks (dying ReLU).

1. **Artificial neural network**
   1. The most common type: layer-wise, and fully connected.
   2. e.g. a 3-layer neural network. (3 = 2 hidden layers + 1 output layer)

ダイアグラム

自動的に生成された説明

* 1. Note we do not counter the input layer.
  2. Activation function for every hidden layer, and a loss function (SVM or Softmax classifier) for output layer.
  3. The output is usually taken to represent the class scores (e.g. in classification), or some kind of real-valued target (e.g. in regression).
  4. How many learnable parameters in this case?

4 + 4 + 1 = 9 neurons

[3 x 4] + [4 x 4] + [4 x 1] = 12 + 16 + 4 = **32 weights**

4 + 4 + 1 = **9 biases**

For a total of **41 learnable parameters**.

* 1. Setting numbers of layer and neuron
     1. More layer and more neuron, better learning of complex data
     2. A large network is generally better than a small network.
     3. Yet higher risk of overfitting: a high accuracy on training data, however low for test data
     4. Solution to overfitting: the regularization strength is the preferred way. e.g.

グラフ, 散布図

自動的に生成された説明

Next study:

[Neural Networks Part 2: Setting up the Data and the Loss](https://cs231n.github.io/neural-networks-2/)

preprocessing, weight initialization, batch normalization, regularization (L2/dropout), loss functions