1 General Formula

Frobenius Norm: $\|\theta\|_F = \sqrt{\sum_j \theta_j^2}$ **L1 Norm**: $\sum_i |\theta_i|$

2 Neurons

2.1 Biology

Neuron contains **Soma, Axon, Dendrites**. Signals travel away from Soma via Axon.

2.1.1 Membrane

Membranes contain Sodium and Potassium Channels and a Sodium-Potassium Pump. Each Channel:

1. Na: Outside of cell
2. K: Inside of cell
3. Na-K Pump: 3 Sodium out for 2 Potassium in

Voltage Difference (Membrane Potential): Difference in voltage on either side of membrane. Resting potential is –70mV. This resting potential is enforced by the Na-K Pump.

Action Potential: Electrical impulse travelling along axon to the synapse.

2.2 Hodgkin-Huxley Model

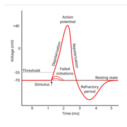
Model of Neuron based on Ion Channel components.

Fraction of K+ channels open is $n(t)^4$. There are four identical gates in K+ channel. Fraction of Na+ channels open is $h(t)m^{(t)^3}$. There are three identical gates and one unique gate in Na+ channel.

Gate dynamic system is a(t): $\frac{da}{dt} = \frac{1}{\tau_{a(V)}}\left(a_{\infty(V)} - a\right)$

Membrane Potential Dynamics: $c\frac{dV}{dt} = J_{in} - g_{L(V-V_L)} - g_{Na}m^3h(V - V_{Na}) - g_K n^4(V - V_k)$ c: Membrane capacitance $I = C\frac{dV}{dt}$: Net current inside cell J_{in}: Input current from other Neurons g_L: Leak conductance (membrane not impenetrable to ions) g_{Na}: Maximum Na conductance g_K: Maximum K conductance

Action Potential form:



Process: Stimulus breaks threshold causing Na+ channels to open, then close at action potential. Potassium channels open at action potential and close at refractory period.

2.3 Leaky-Integrate-and-Fire Model

Spike shape is constant over time, more important to know when spiked than shape. LIF only considers sub-threshold voltage and when peaked.

Dynamics system: $c\frac{dV}{dt} = J_{in} - g_{L(V-V_L)} \Rightarrow \tau_m\frac{dV}{dt} = V_{in} - (V - V_L)$ for $V < V_{th}$

This is dimensioned. Dimensionless converts $v_{in} = \frac{v_{in}}{v_{th} - V_L}$ to become $\tau_m\frac{dv}{dt} = v_{in} - v$

Then spike occurs when $v = 1$ and we set a refractory period of t_{ref} before starting at 0 again.

Explicit Model: $v(t) = v_{in}\left(1 - e^{-\frac{t}{\tau}}\right)$

Firing Rate: $\frac{1}{\tau_{\text{ref}} - \tau_m \ln(1 - 1v_{\in})}$ for $v_{\in} > 1$ **Tuning Curve**: Graph showing how neuron reacts to different input currents.

2.4 Activation Functions (Sigmoidal)

Logistic Curve: $\frac{1}{1+e^{-z}}$ Arctan: $\arctan(z)$ Hyperbolic Tangent: $\tanh(z)$ Threshold: 0 if $z < 0$; 1 if $z \geq 0$ Rectified Linear Unit: $\max(0, z)$ Softplus: $\log(1 + e^z)$

2.4.1 Multi-Neuron Activation Functions

SoftMax: $\frac{\exp(z_i)}{\sum_j \exp(z_j)}$ Converts elements to probability distribution (sum to 1) and normalizes values ArgMax: Largest element remains nonzero, everything else 0

3 Synapses

In real neuron, **Presynaptic action potential** releases **neurotransmitters** across **synaptic cleft** which binds to **postsynaptic** receptors

Equation for current entering **postsynaptic neuron**: $h(t) = \begin{cases} kt^n e^{-\frac{t}{\tau_s}} \text{ if } t \geq 0 (\text{for some } n \in \mathbb{Z}_{\geq 0}) \\ 0 \text{ if } t < 0 \end{cases}$

k is selected so $\int_0^\infty h(t)dt = 1 \Rightarrow k = \frac{1}{n!\tau_s^{n+1}}$

Postsynaptic potential filter: $h(t)$

Spike train: Series of multiple spikes $a(t) = \sum_{p=1}^k \delta(t - t_p)$

Dirac function: Infinite at $t = 0$, 0 everywhere else. Properties: $\int_{-\infty}^\infty \delta(t)dt = 1$, $\int_{-\infty}^\infty f(t)\delta(t - \tau)dt = f(\tau)$

Filtered Spike Train: $s(t) = a(t) * h(t)$ For each spike in spike train, run the postsynaptic potential filter on it, then sum each spike. Essentially, convolve the spike train to the postsynaptic potential filter.

Derivation:

$$s(t) = a(t) * h(t) = \int \sum_p \delta(t - t_p)h(t - \tau)d\tau = \sum h(t - t_p)$$

3.1 Neuron Activities

Neurons have multiple connections, with different strengths (weights). We can represent weights between neuron layers via weight matrix: $W \in R^{N \times M}$

Compute neuron function as: $\vec{z} = \vec{x}W + \vec{b}$, thus $\vec{y} = \sigma(\vec{z})$

3.1.1 Bias Representation

Add a neuron with constant value 1 for each layer, and use its weights as biases.

$\vec{x}W + \vec{b} = (\vec{x} \;\; 1) \cdot \begin{pmatrix} W \\ \vec{b} \end{pmatrix}$

3.1.2 Connections between spiking Neurons

Let $n = 0$ for simplicity for $h(t)$, then it is a solution of $\tau_s\frac{ds}{dt} = -s, s(0) = \frac{1}{\tau_s}$

3.2 Full LIF Model

Dynamics are described by: $\begin{cases} \tau_m\frac{dv_i}{dt} = s_i - v_i \text{ if not in refractory period} \\ \tau_s\frac{ds_i}{dt} = -s_i \end{cases}$

If v_i reaches 1, then start refractory period, send spike, reset v_i to 0. If spike arrives from neuron j, then $s_i \leftarrow s_i + \frac{w_{ij}}{\tau_s}$

4 Neural Learning

4.1 Supervised Learning

Desired output is known, and we can minimize error within our model's predictions

Regression: Output is continuous-based function of inputs, goal is to get closest to output function.

Classification: Outputs are in discrete categories.

4.2 Unsupervised Learning

Output is not known, so goal is to find efficient structure of input

4.3 Reinforcement Learning

Feedback is given, but it's uninformative, and depends on lots of variables (ex. Chess AI)

4.4 Optimization

Given neural model, goal is to optimize weights to minimize loss function.

$\min_\theta \mathbb{E}_{(x,t) \in \text{ data}}[\mathcal{L}(f(x;\theta), t(x))]$

5 Universal Approximation Theorem

For all continuous functions in the domain of n parameters in $[0, 1]$ domain each, can be approximated as finite sums of sigmoid functions.

Sigmoid Function: Goes to 1 for positive infinity and 0 for negative infinity.

5.1 Process

1. By giving infinite weight to $\sigma(wx)$, this approximates a threshold function.
2. We create piece function as difference of threshold functions: $P(x; b, \delta) = H(x; b) - H(x; b + \delta)$
3. Approximate each section of the function, as $G(x) = \sum_{j=1}^{N'} f(a_j)P(x; b_j, \delta_j)$ each is within ε band

6 Loss functions

Single Error: $L(y, t)$ is error between one output y and target t Dataset Error: $\mathbb{E} = \frac{1}{N}\sum_{i=1}^N L(y_i, t_i)$ as average error over entire dataset.

6.1 Mean Squared Error

Single Error: $L(y, t) = \frac{1}{2}\|y - t\|_2^2$ Activation Function: Linear activation functions (ReLU) Problems: linear regression problems.

6.2 Bernoulli Cross Entropy

Single Error: $L(y, t) = -\log(P(y, t)) = -\log(t^y(1 - y)^{1-t}) = -(t\log(y) + (1 - t)\log(1 - y))$ Activation Function: Logistic function Problems: Output values are in range $[0, 1]$;

6.3 Categorical Cross Entropy

Single Error: $L(y, t) = -\log(P(x in C_k, t)) = -\log\left(\prod_{k=1}^K (y^k)^{t^k}\right) = -\sum_{k=1}^k t^k \log(y^k)$ Activation Function: Softmax Problems: Classification problems with one-hot vectors

7 Gradient Descent

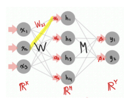
Our goal is to minimize $E(\theta)$ (expected Error), so we define gradient $\nabla_\theta E = \left(\frac{\partial E}{\partial \theta_1} \;\; \cdots \;\; \frac{\partial E}{\partial \theta_n}\right)$

Error gradient: $\tau\frac{d\theta}{dt} = \nabla E(\theta)$, t is a parameter for "time" as we move through parameter space. Euler step: $\theta_{n+1} = \theta_n + k\nabla E(\theta_n)$

8 Error Backpropagation

$\nabla_{z^{(l+1)}}E = \frac{\partial E}{\partial z^{(l+1)}}$ $h^{(l+1)} = \sigma(z^{(l+1)}) = \sigma(W^{(l)}h^{(l)} + b^{(l+1)})$ Basically, h is hidden layer, z is input current, W is weight matrix, b is bias. $\nabla_{z^{(l)}}E = \frac{dh^{(l)}}{dz^{(l)}} \odot \left[\nabla_{z^{(l+1)}E \cdot (W^{((l))^T})}\right]$ $\odot$ is hadamard product, which does element by element multiplication. We transpose because W_{ij} is connection from ith node in l to jth node in $l + 1$

Note that $\vec{a} = \vec{x}W$ in this diagram:



$\frac{\partial E}{(\partial W_{ij}^{(l)} = \frac{\partial E}{\partial z_j^{(l+1)}} \cdot \frac{\partial z_j^{(l+1)}}{\partial W_{ij}^{(l)}} = \frac{\partial E}{\partial z_j^{(l+1)}} \cdot h_i^{(l)}}$

Finally, $\nabla_{z^{(l)}}E = \sigma'(z^{(l)}) \odot \left(\nabla_{z^{(l+1)}}E \cdot (W^{(l)})^T\right)$ $\nabla_{W^{(l)}}E = [h^{(l)}]^T \nabla_{z^{(l+1)}}E$

8.1 Vectorization

We can generalize this process to take a batch of samples by letting x be a matrix of samples instead of just one sample. Then, note $\nabla_{z^{(l)}}E$ is a matrix with same dimension as $z^{(l)}$ as desired. Further, note that $\nabla_{W^{(l)}}E$ is a gradient vector that sums the weight gradient matrix from each sample.

9 Auto-Differentiation

9.1 Expression Graph

Each operation is a square with its variable dependencies. Each variable has a pointer to its creator, which is the operation that created it.

9.1.1 Pseudocode

Function $f = g(x, y, \ldots)$
- Create g Op object
- Save references to args $x, y, \ldots$
- Create Var for output f
- Set g.val as $g(x, y, \ldots)$
- Set f.creator to the g Op.

9.2 Differentiate

With each object in our graph, store derivative of total expression with respect to itself in member *grad* Use chain rule with parent operation Op.grad to get current grad. Ex: If y is parent of x, then x.grad = y.grad $\cdot \frac{\partial y}{\partial x}$ Also add wherever multiple branches converge, as is normal in derivatives calculation.

Backward method:

```
class Var:
  def backward(s):
    self.grad += s
    self.creator.backward(s)

class Op:
  def backward(s):
    for x in self.args:
      x.backward(s * partialDeriv(Op, x))
```

In Var, self.val, self.grad, s must have same shape. In Op, s must be shape of operation output

10 Neural Nets w/w Auto-Diff

10.1 Gradient Descent Pseudocode

- Initialize v, κ
- Make expression graph for E
- Until convergence:
  - Evaluate E at v
  - Zero-grad
  - Calculate gradients
  - Update $v \leftarrow v - \kappa$ v.grad

10.2 Neural Learning

Optimizing our weights and biases for our loss function. $W \longleftrightarrow -\kappa\nabla_W E$ By making network with AD classes, we leverage backward() to optimize gradient computation.

10.2.1 Pseudocode

Given Dataset (X, T) and network model **net**, with parameters θ and loss function L
- y = **net**(X)
- loss = L(y, T)
- loss.zero_grad()
- loss.backward()
- $\theta \leftarrow \kappa \cdot \theta.\text{grad}$

11 Overfitting

If big discrepancy in accuracy between training and testing, then we're "overfitting"

11.1 Problems

If training is too small, it's overfitting. If test error is much bigger than training error.

11.1.1 Solutions

Validation: Keep subsection of testing error as validation error to determine proper hyperparameters, then finally we test to determine the correctness. **Regularization by Weight Decay**: Expand error to worry about hyperparameters as well. $\tilde{E}(\hat{y}, t; \theta) = E(\hat{y}, t; \theta) + \frac{\lambda}{2}\|\theta\|_F^2$ Then new rules are $\nabla_{\theta_i}\tilde{E} = \nabla_{\theta_i}E + \lambda\theta_i$ and $\theta_i \leftarrow -\kappa\nabla_{\theta_i}E - \kappa\lambda\theta_i$ **Data Augmentation**: Add slightly modified versions of data to make more samples **Dropout**: Drop some random nodes to distribute computation.

12 Optimization

12.1 Stochastic Gradient Descent

Take random batch of samples, then take the expected value over all of them: $E(\tilde{y}, \tilde{\tau}) = \frac{1}{B}\sum_{d=1}^B (y_{\gamma_d}, t_{\gamma_d})$

12.2 Momentum

Usecases: When gradient descent oscillates very often before converging to a local minimum; optimization stops at shallow local optimum without global optimum. Gradient is a force instead of slope. $\theta_{n+1} = \theta_n + \Delta t v_n$ and $v_{n+1} = (1 - r)v_n + \Delta t A_n$ where A_n represents the gradient vector, so we make $v^{(t)} \leftarrow \beta v^{(t-1)} + \nabla_W E$ and $w^{(t)} \leftarrow w^{(t-1)} - \eta v^{(t)}$