Neural Network: Learning and Evaluation

Jacob Wahbeh and Roei Burstein

Overview

There are several steps to take in order to train a neural network:

- Gradient Check (Avoid pitfalls)
- Babysitting the learning process
  - Loss function, Train/val accuracy, etc
- Parameter Updates
  - Implementation
- Evaluation
  - Creating an ensemble

Gradient Checks

There are two ways to compute the gradient:

- Numerical: Slower and less exact, but easier (requires no calculus)
- Analytic: Faster and more exact, but prone to errors (requires calculus)

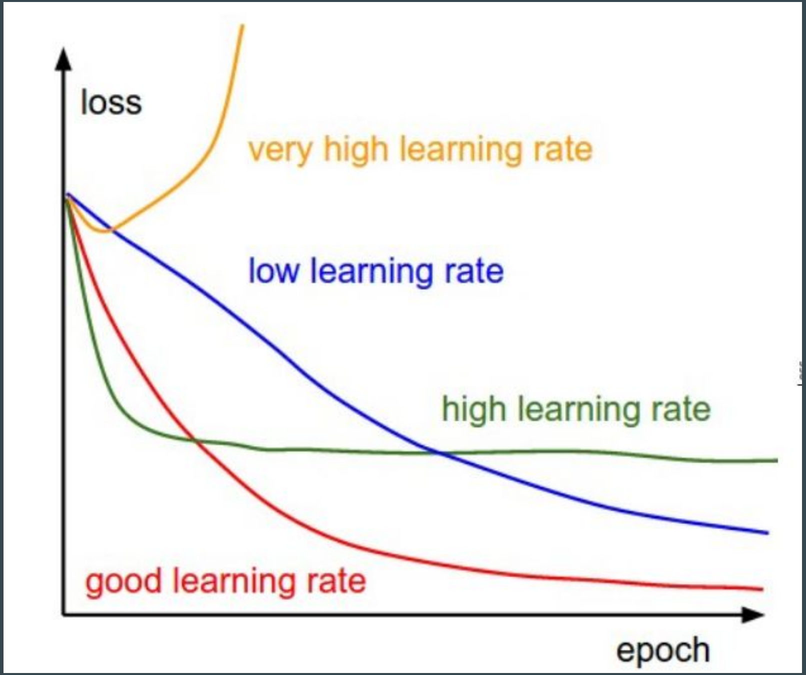
Relative Error:

- Less than 10^-7 is good
- Small batch of data
- Be aware of Pitfalls

$$\frac{|f'_a - f'_n|}{\max(|\ f'_a\ |, |\ f'_n\ |)}$$

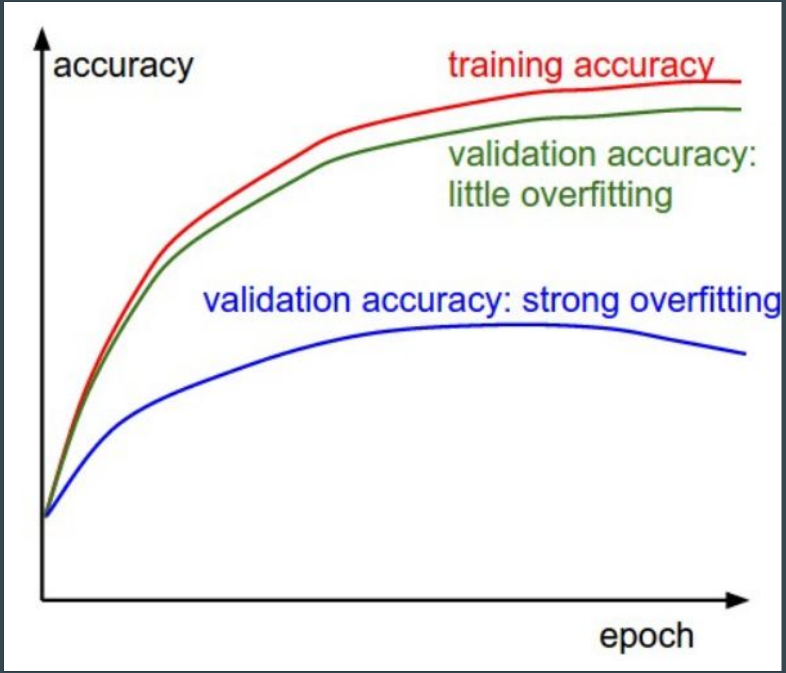
Babysitting the Learning Process

- Essential for choosing hyperparameters
- Tracking the loss function
- Too high of learning rate can be chaotic
- Parameters bouncing around, can't settle
- Too low would be linear progression
- Want loss to be as close to zero

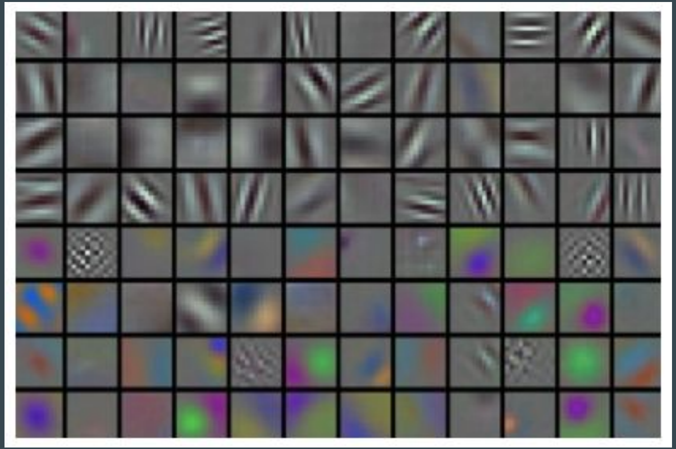


Training/Validation Accuracy

- When training a classifier, it is important to consider the difference between the training and validation accuracy.
- Look at the gap between green/red curves and blue/red curves.
- If the validation accuracy is far from training accuracy, this means the model is overfitted to the data



More Learning Process Checks



- Ratio of Weights: update vs value
  - Should be around 10^-3
- Gradient distribution per layer
  - Tanh neurons should see even range [-1. 1]
- First-layer visualization
  - When working with pixels

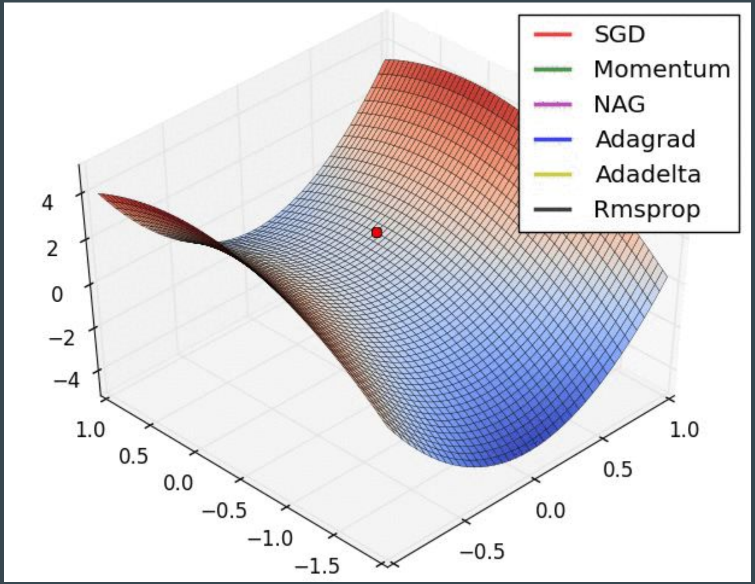
Parameter Updates

- After the gradient is computed using backward propagation, each parameter can be updated using several levels of complexity:
  - Vanilla Update (red)
  - Momentum Update (green)
  - Nesterov Momentum (purple)

Adagrad, adadelta, and rmsprop are

second order methods of updating

Parameters and involve more complex methods



Methods of Updating Parameters

Vanilla Update:

```
# Vanilla update
x += - learning_rate * dx
```

- Basic update to parameters. Update with negative gradient

Momentum Update:

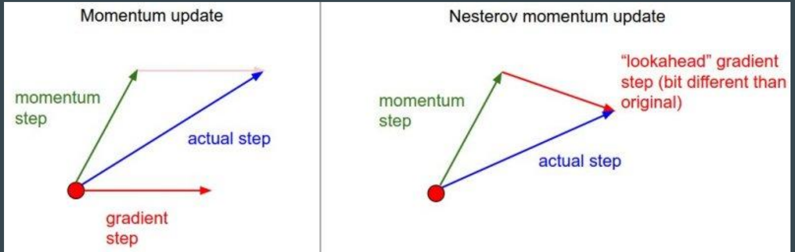
```
# Momentum update
v = mu * v - learning_rate * dx # integrate velocity
x += v # integrate position
```

- Update that behaves similar to potential energy with random starting point

Nesterov Momentum:

```
x_ahead = x + mu * v
# evaluate dx_ahead (the gradient at x_ahead instead of at x)
v = mu * v - learning_rate * dx_ahead
x += v
```

- Similar to Momentum Update, but computes gradient step with “lookahead”



Annealing Learning Rate

It is important to know when/how to decay the learning rate so that the model eventually settles:

- Step Decay (Reduce by some factor every few generations)
- Exponential Decay (Reduce exponentially) $\alpha = \alpha_0 e^{-kt}$,
- 1/t Decay (Reduce hyperbolically) $\alpha = \alpha_0/(1 + kt)$

For the above formulas, α_0, k are hyper parameters and t is the iteration number

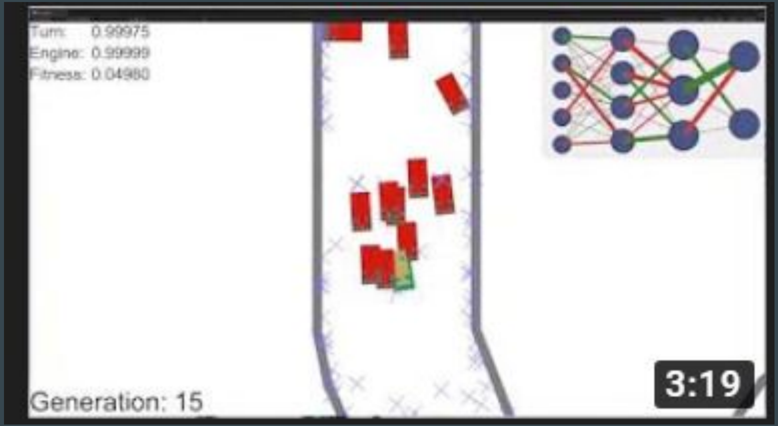
Hyperparameter optimization

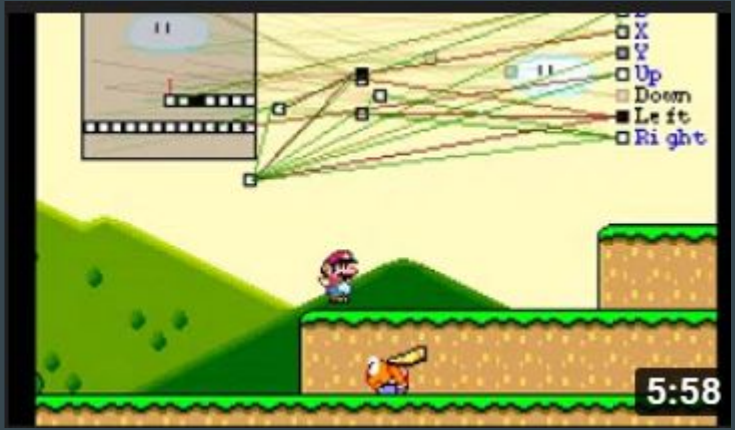
3 Common hyperparameters

- Initial learning rate
- Learning rate decay schedule
- Regularization strength

Implementation

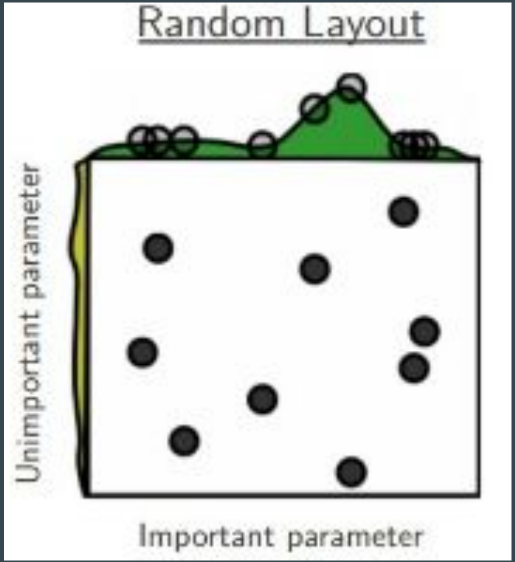
- Worker randomly samples hyperparameters
  Performs optimization on the individual
- Master sorts, kills, and reproduces workers
  Inspect checkpoints and plot their training
- Worker is the car, Master chooses which car
  Gets passed to next generation





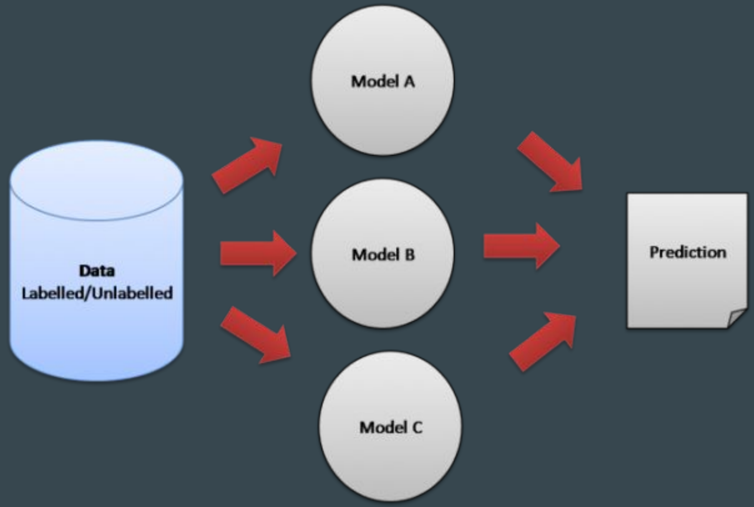
Hyperparameter checks

- Search on log scale `learning_rate = 10 ** uniform(-6, 1)`
- Prefer random search, easier to implement and better optimization
- Values on border means change range
- Start wide and narrow as time goes on



Evaluation

Forming an ensemble of different models and averaging their predictions can improve the performance.



- Same Model, Different Initializations:
    - Initialize parameters of each model with different values
    - Possible danger is that only difference in model is initial values of hyperparameters
- Top Models Discovered During Cross-Validation
    - Use cross-validation to determine the best hyperparameters
    - Pick top few to form the ensemble
    - Builds off first method by choosing only those with best hyperparameters for averaging

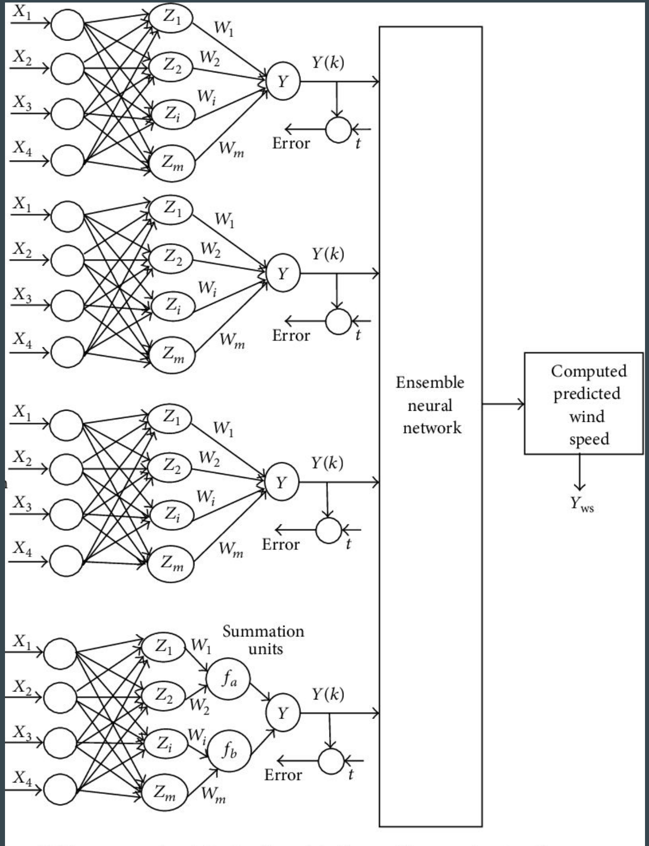
More Ensemble models

Different checkpoints of a single model

- If training is expensive, checkpoints overtime
- At each epoch, save the parameters
- Average predictions at end of total training

Running average of parameters during training

- Save a second copy of the weights maintains a decaying sum of previous weights
- Average over many iterations as you train
- Achieves better validation error



URE 2: The proposed architectural model of ensemble neural network

Summary

In order to train a neural network:

- Gradient check your implementation
- Monitor training/validation accuracy
- Update parameters (SGD, Momentum, Nesterov)
- Decay learning rate over time
- Search for good hyperparameters randomly
- Form model ensemble

Thank you!

Questions?

Works Cited

- http://cs231n.github.io/neural-networks-3/#summary