Neural Networks: Practical Concerns

Machine Learning

THE UNIVERSITY OF UTAH

Based on slides and material from Geoffrey Hinton, Richard Socher, Dan Roth, Yoav Goldberg, Shai Shalev-Shwartz and Shai Ben-David, and others

Neural Networks

- What is a neural network?
- Predicting with a neural network
- Training neural networks
- Practical concerns

This lecture

- What is a neural network?
- Predicting with a neural network
- Training neural networks
- Practical concerns

Training neural networks with SGD

- **No guarantee** of convergence, may oscillate or reach a local minima
- In practice, many large networks are trained on **large amounts of data** for realistic problems

Training neural networks with SGD

- **No guarantee** of convergence, may oscillate or reach a local minima

- In practice, many large networks are trained on **large amounts of data** for realistic problems

- Many epochs (sometimes thousands) may be needed for adequate training
  - Large data sets may require hours/days/weeks of CPU or GPU time!
  - Sometimes specialized hardware even

Training neural networks with SGD

- **No guarantee** of convergence, may oscillate or reach a local minima
- In practice, many large networks are trained on **large amounts of data** for realistic problems
- Many epochs (sometimes thousands) may be needed for adequate training
  - Large data sets may require hours/days/weeks of CPU or GPU time!
  - Sometimes specialized hardware even
- **Termination criteria**: Number of epochs, Threshold on training set error, No decrease in error, Increased error on a validation set

Training neural networks with SGD

- **No guarantee** of convergence, may oscillate or reach a local minima
- In practice, many large networks are trained on **large amounts of data** for realistic problems
- Many epochs (sometimes thousands) may be needed for adequate training
  - Large data sets may require hours/days/weeks of CPU or GPU time!
  - Sometimes specialized hardware even
- **Termination criteria**: Number of epochs, Threshold on training set error, No decrease in error, Increased error on a validation set
- To **avoid local minima**: several trials with different random initial weights with majority or voting techniques

Minibatches

- Stochastic gradient descent:
  - Take a random example at each step
  - Write down the loss function with that example
  - Compute gradient this loss and take a step

Why should we take only one random example at each step?

Minibatches

- Stochastic gradient descent:
  - Take a random example at each step
  - Write down the loss function with that example
  - Compute gradient this loss and take a step

Why should we take only one random example at each step?

- Stochastic gradient descent with minibatches:
  - Collect a small number of random examples (the minibatch) at each step
  - Write down the loss function with that example
  - Compute gradient this loss and take a step

Minibatches

- Stochastic gradient descent:
  - Take a random example at each step
  - Write down the loss function with that example
  - Compute gradient this loss and take a step

Why should we take only one random example at each step?

- Stochastic gradient descent with minibatches:
  - Collect a small number of random examples (the minibatch) at each step
  - Write down the loss function with that example
  - Compute gradient this loss and take a step

- New hyperparameter: The size of a minibatch
  - Often governs how fast the learning converges
  - Hardware considerations around memory could dictate how big the minibatch could be

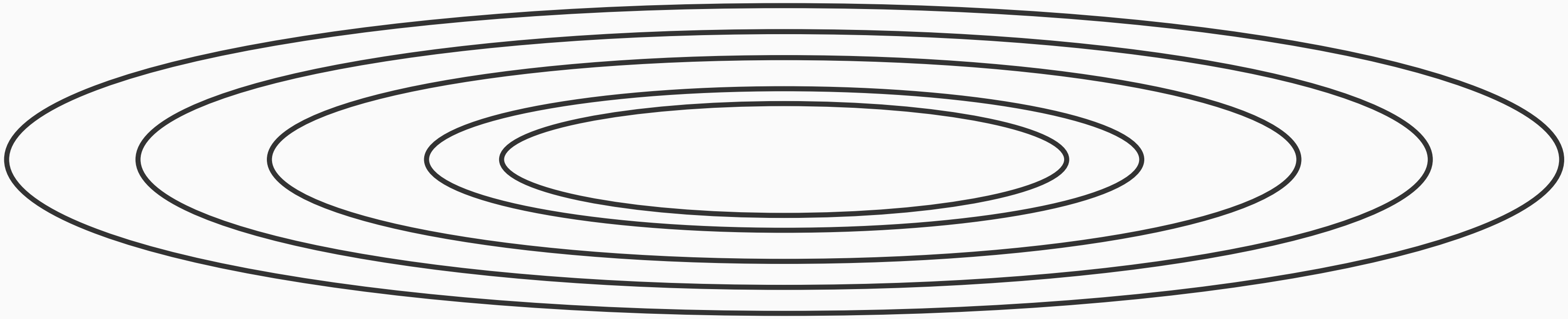
Gradient tricks

Simple gradient descent updates the parameters using the gradient of one example (or a minibatch of them), denoted by g_i

$$\text{parameters} \leftarrow \text{parameters} - \eta g_i$$

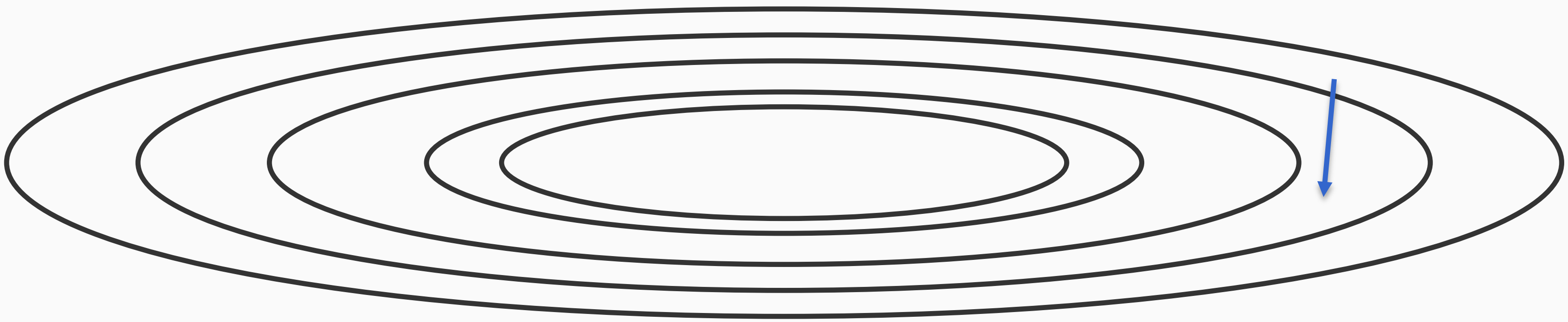
Gradient tricks

Gradients could change much faster in one direction than another



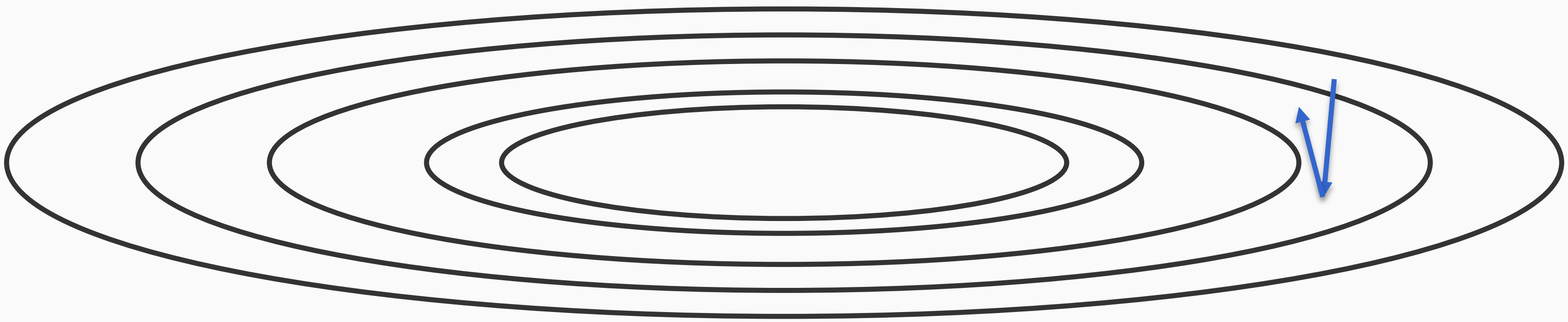
Gradient tricks

Gradients could change much faster in one direction than another



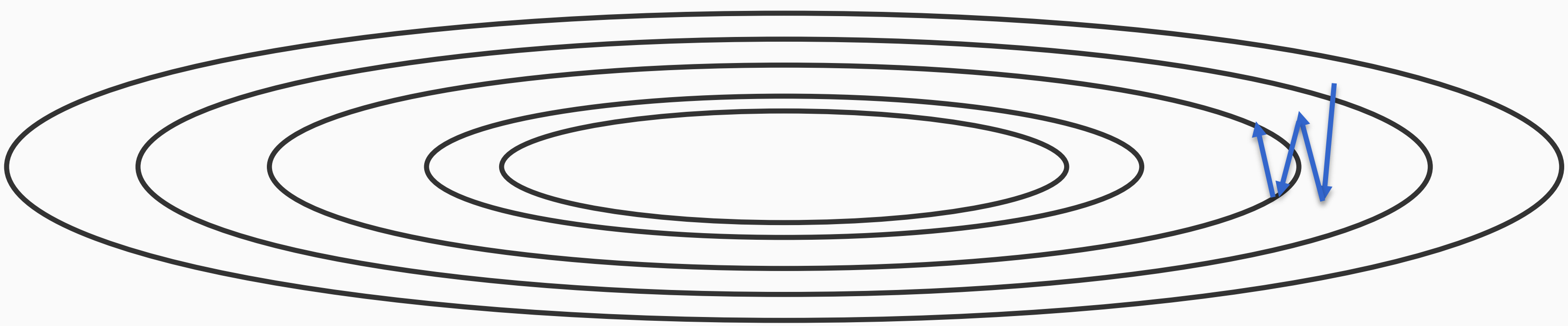
Gradient tricks

Gradients could change much faster in one direction than another



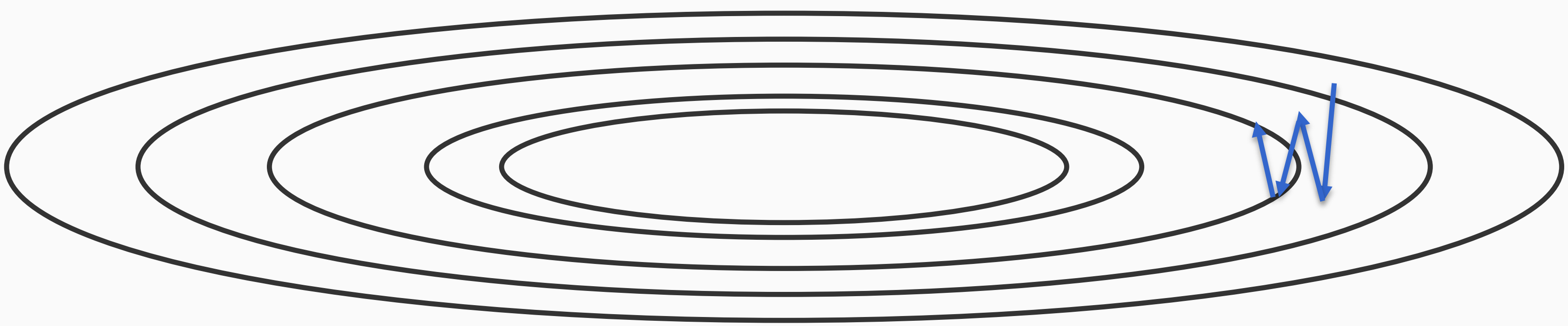
Gradient tricks

Gradients could change much faster in one direction than another



Gradient tricks

Gradients could change much faster in one direction than another



When gradients change very fast, this could make learning either slow, or worse, unstable.

The quality of the model could change drastically based on how many epochs you run

Gradient tricks: Momentum

Instead of updating with the gradient (g_i), use a moving average of gradients ($\mathbf{v}_t$) to update the model parameters

- In the inner loop:
  - $\mathbf{v}_t \leftarrow \mu \mathbf{v}_{t-1} + \eta_t g_i$
  - $\text{parameters} \leftarrow \text{parameters} - \mathbf{v}_t$

Gradient tricks: Momentum

Instead of updating with the gradient (g_i), use a moving average of gradients ($\mathbf{v}_t$) to update the model parameters

- In the inner loop:
  - $\mathbf{v}_t \leftarrow \mu \mathbf{v}_{t-1} + \eta_t g_i$
  - $\text{parameters} \leftarrow \text{parameters} - \mathbf{v}_t$

Update = average of previous update $\mathbf{v}_{t-1}$ and the gradient

Gradient tricks: Momentum

Instead of updating with the gradient (g_i), use a moving average of gradients ($\mathbf{v}_t$) to update the model parameters

- In the inner loop:
  - $\mathbf{v}_t \leftarrow \mu \mathbf{v}_{t-1} + \eta_t g_i$
  - $\text{parameters} \leftarrow \text{parameters} - \mathbf{v}_t$

Update = average of previous update $\mathbf{v}_{t-1}$ and the gradient

The hyperparameter μ controls how much of the previous update should be retained. Typical value $\mu = 0.9$

Gradient tricks: Momentum

Instead of updating with the gradient (g_i), use a moving average of gradients ($\mathbf{v}_t$) to update the model parameters

– In the inner loop:
  - $\mathbf{v}_t \leftarrow \mu \mathbf{v}_{t-1} + \eta_t g_i$
  - $\text{parameters} \leftarrow \text{parameters} - \mathbf{v}_t$

Update = average of previous update $\mathbf{v}_{t-1}$ and the gradient

The hyperparameter μ controls how much of the previous update should be retained. Typical value $\mu = 0.9$

Momentum smooths out the updates by using a weighted average of all previous gradients at each step

Gradient tricks: AdaGrad, RMSProp, Adam

- AdaGrad: Each parameter has its own learning rate. If $g_{i,t}$ is the gradient for the i^{th} parameter at step t,

$$c_i \leftarrow c_i + g_{i,t}^2$$

$$\text{parameters}_i \leftarrow \text{parameters}_i - \frac{\eta}{\alpha + \sqrt{c_i}} g_{i,t}$$

Gradient tricks: AdaGrad, RMSProp, Adam

- AdaGrad: Each parameter has its own learning rate. If $g_{i,t}$ is the gradient for the i^{th} parameter at step t,

$$c_i \leftarrow c_i + g_{i,t}^2$$

$$\text{parameters}_i \leftarrow \text{parameters}_i - \frac{\eta}{\alpha + \sqrt{c_i}} g_{i,t}$$

- RMSProp: Similar to AdaGrad, but more recent gradients are weighted more in the denominator

$$c_i \leftarrow \Delta \cdot c_i + (1 - \Delta) \cdot g_{i,t}^2$$

Gradient tricks: AdaGrad, RMSProp, Adam

- AdaGrad: Each parameter has its own learning rate. If $g_{i,t}$ is the gradient for the i^{th} parameter at step t,

$$c_i \leftarrow c_i + g_{i,t}^2$$

$$\text{parameters}_i \leftarrow \text{parameters}_i - \frac{\eta}{\alpha + \sqrt{c_i}} g_{i,t}$$

- RMSProp: Similar to AdaGrad, but more recent gradients are weighted more in the denominator

$$c_i \leftarrow \Delta \cdot c_i + (1 - \Delta) \cdot g_{i,t}^2$$

- Adam: A combination of many ideas:
  - Momentum to smooth gradients
  - RMSProp like approach for adaptively choosing learning rate with more recent gradients being weighted higher
  - Additional terms to avoid bias introduced during early gradient estimates
  - Currently the most commonly used variant of gradient based learning

Preventing overfitting

- Running too many epochs may *over-train* the network and result in over-fitting

Preventing overfitting

- Running too many epochs may *over-train* the network and result in over-fitting
- Keep a **hold-out validation set** and test accuracy after every epoch

Preventing overfitting

- Running too many epochs may *over-train* the network and result in over-fitting
- Keep a **hold-out validation set** and test accuracy after every epoch
- Maintain weights for best performing network on the validation set and return it when performance decreases significantly beyond that

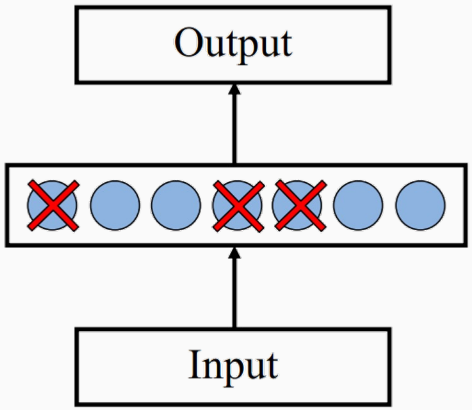
Preventing overfitting

- Running too many epochs may *over-train* the network and result in over-fitting
- Keep a **hold-out validation set** and test accuracy after every epoch
- Maintain weights for best performing network on the validation set and return it when performance decreases significantly beyond that
- To avoid losing training data to validation:
  - Use k-fold cross-validation to determine the average number of epochs that optimizes validation performance
  - Train on the full data set using this many epochs to produce the final results

Avoiding overfitting with Dropout training

Hinton et al, 2012

- During training, for each step, decide whether to delete a hidden unit with some probability *p*
  - That is, make predictions using only a randomly chosen set of neurons
  - Update only these neurons



- Tends to avoid overfitting

- Has a model averaging effect
  - Only some parameters get trained at any step

Number of hidden units

- **Too few hidden units** prevent the system from adequately fitting the data and learning the concept.

- **Using too many hidden units** leads to over-fitting.

- Cross-validation or performance on a held out set can be used to determine an appropriate number of hidden units.

Neural networks: What we saw

- What is a neural network?
  - Multiple layers
    - Inner layers learn a **representation** of the data
  - Highly expressive
    - Is this always a good thing? What about the VC dimension? Overfitting?

- Training neural networks
  - Backpropagation

What we did not see

Vast area, fast moving

- Many new models, algorithms and tricks for learning that tweak on the basic gradient method
- Massively growing models and datasets

Some named neural networks

- Restricted Boltzmann Machines and autoencoders: Learn a latent representation of the data
- Convolutional neural network: Modeled after the mammalian visual cortex, currently the state of the art for object recognition tasks
- Recurrent neural networks and Transformers: encode and predict sequences
- Attention: Use a neural network to decide what parts of a set of features are relevant and create an aggregate “attended” representation
- ...And many many more