Machine Learning Tricks

Philipp Koehn

8 October 2024

Machine Learning

- Myth of machine learning

  - given: real world examples
  - automatically build model
  - make predictions

Machine Learning

- Myth of machine learning

  - given: real world examples
  - automatically build model
  - make predictions

- Promise of deep learning

  - do not worry about specific properties of problem
  - deep learning automatically discovers the feature

Machine Learning

- Myth of machine learning

  - given: real world examples
  - automatically build model
  - make predictions

- Promise of deep learning

  - do not worry about specific properties of problem
  - deep learning automatically discovers the feature

- Reality: bag of tricks

Today's Agenda

- No new translation model

- Discussion of failures in machine learning

- Various tricks to address them

Fair Warning

- At some point, you will think:

  Why are you telling us all this madness?
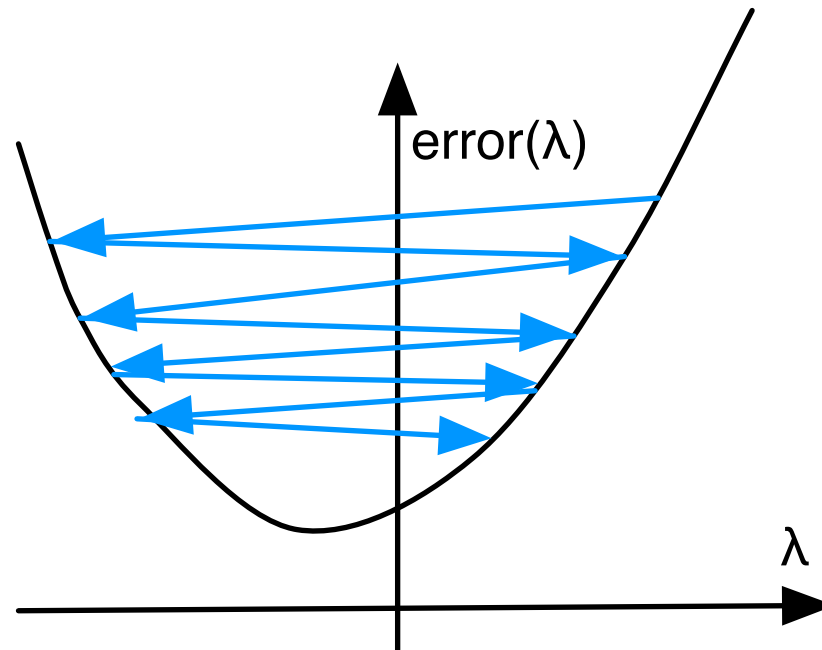
Fair Warning

- At some point, you will think:

  Why are you telling us all this madness?

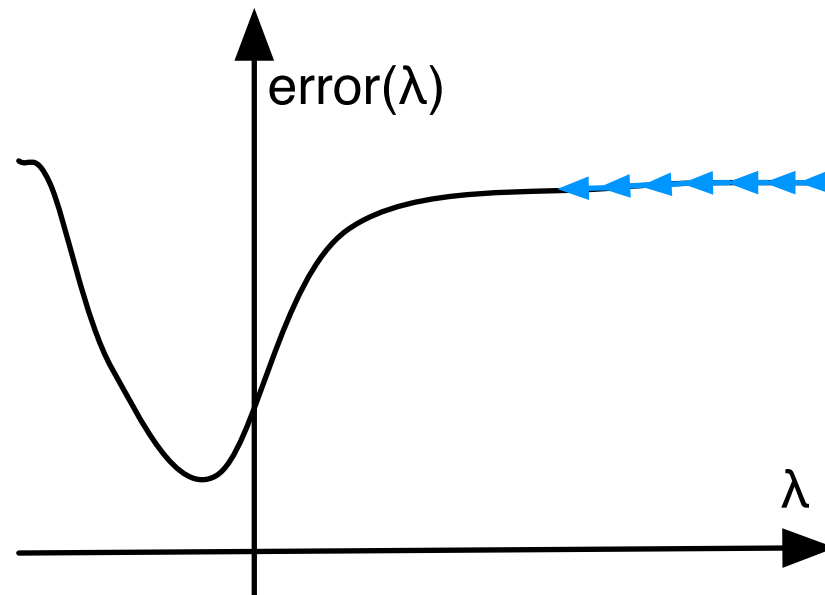- Because pretty much all of it is commonly used

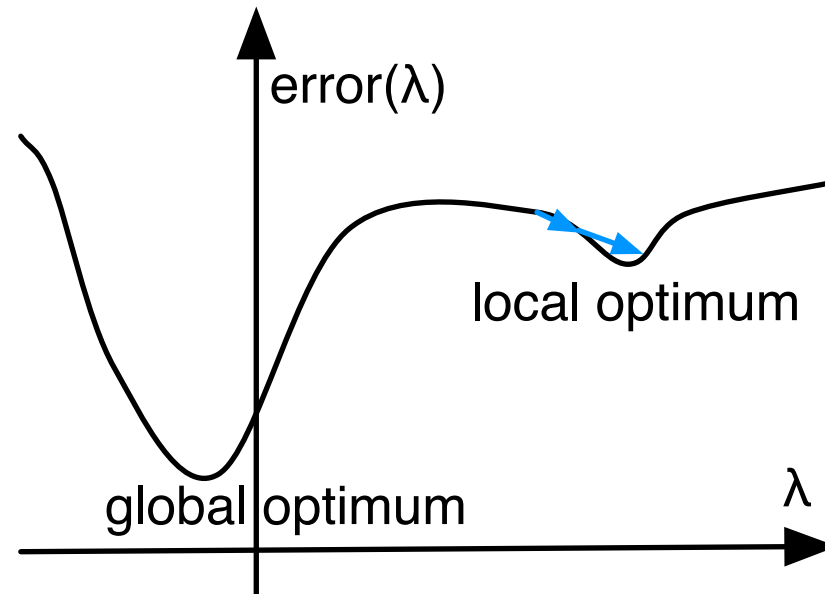# failures in machine learning

Failures in Machine Learning

Too high learning rate may lead to too drastic parameter updates

$\rightarrow$ overshooting the optimum

Failures in Machine Learning

Bad initialization may require many updates to escape a plateau

Local optima trap training

Learning Rate

- Gradient computation gives direction of change

- Scaled by learning rate

- Weight updates

Learning Rate

- Gradient computation gives direction of change

- Scaled by learning rate

- Weight updates

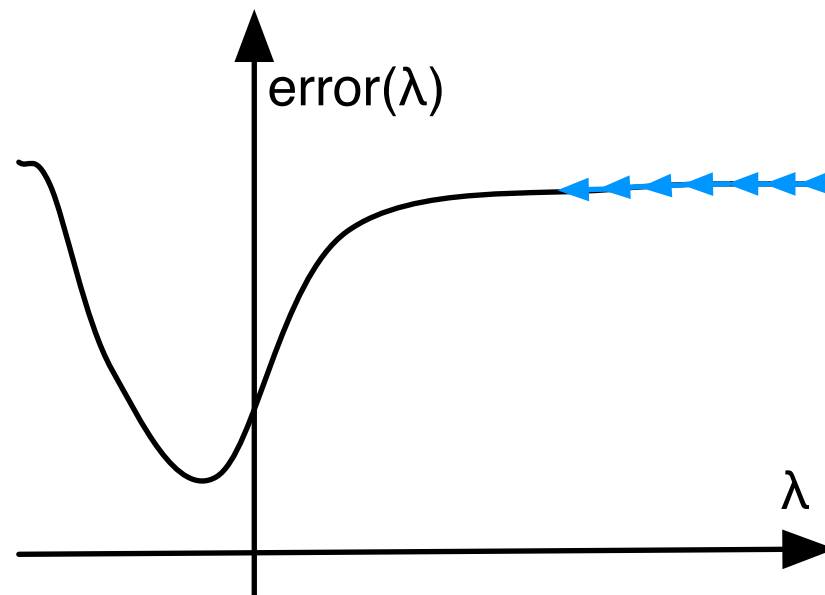- Simplest form: fixed value

- Gradient computation gives direction of change

- Scaled by learning rate

- Weight updates

- Simplest form: fixed value

- Annealing

  - start with larger value (big changes at beginning)
  - reduce over time (minor adjustments to refine model)
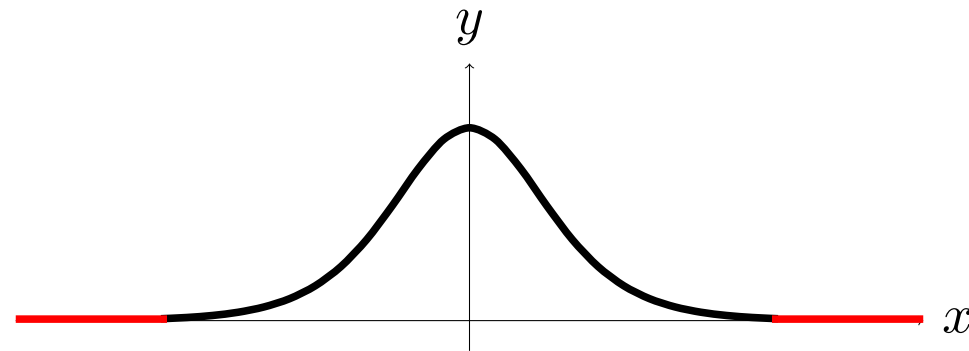
Initialization of Weights

- Initialize weights to random values

- But: range of possible values matters

Sigmoid Activation Function

Derivative of sigmoid

Near zero for large positive and negative values

Rectified Linear Unit
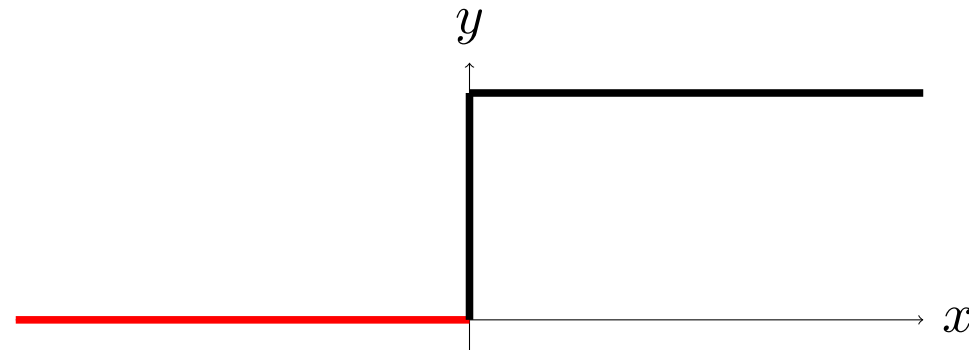
Derivative of ReLU

Flat and for large interval: Gradient is 0

"Dead cells" elements in output that are always 0, no matter the input

- Cartoon depiction
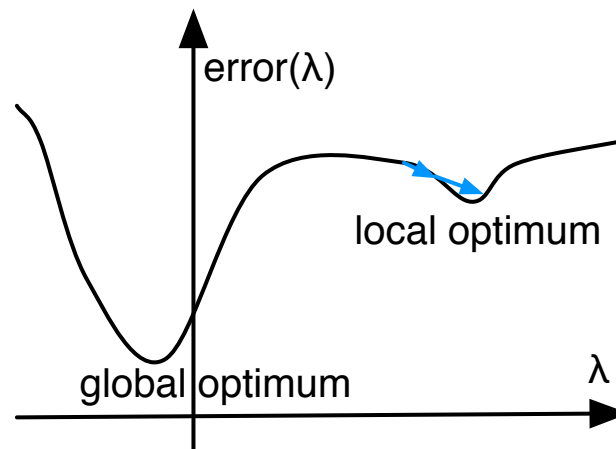
- Reality

  - highly dimensional space
  - complex interaction between individual parameter changes
  - "bumpy"
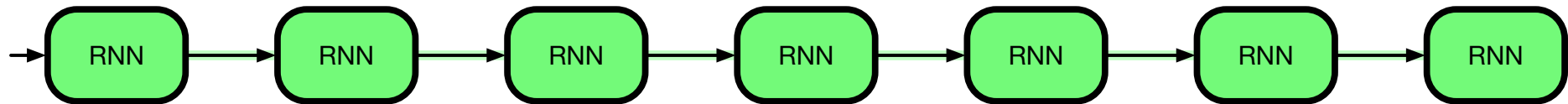
- Repeated multiplication with same values

- If gradients are too low $\rightarrow 0$

- If gradients are too big $\rightarrow \infty$

Overfitting and Underfitting
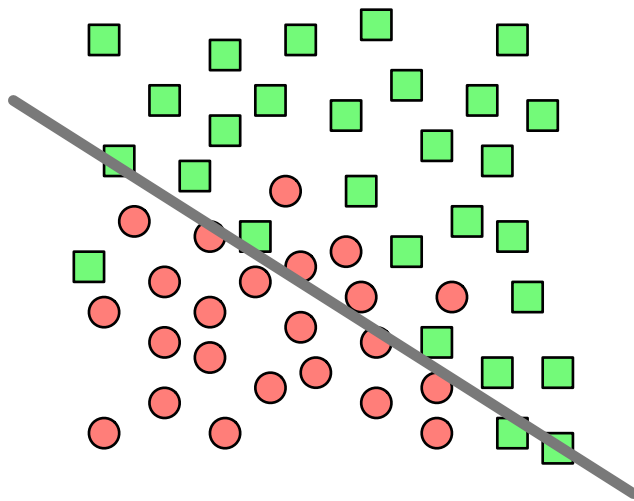
Under-Fitting   **Good Fit**   **Over-Fitting**
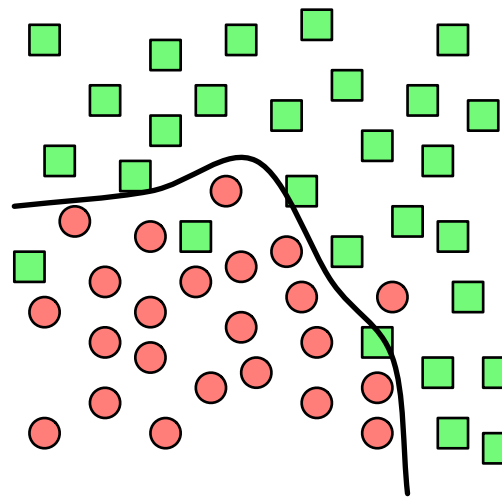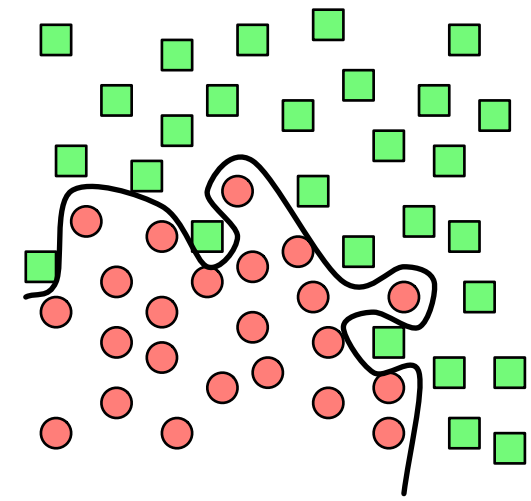
- Complexity of the problem has too match the capacity of the model

- Capacity $\simeq$ number of trainable parameters

ensuring randomness

Ensuring Randomness

- Typical theoretical assumption

  independent and identically distributed

  training examples

Ensuring Randomness

- Typical theoretical assumption

  independent and identically distributed

  training examples

- Approximate this ideal

  – avoid undue structure in the training data
  – avoid undue structure in initial weight setting

- Typical theoretical assumption

    independent and identically distributed

    training examples

- Approximate this ideal

    – avoid undue structure in the training data
    – avoid undue structure in initial weight setting

- ML approach: Maximum entropy training

    – Fit properties of training data
    – Otherwise, model should be as random as possible
      (i.e., has maximum entropy)

Shuffling the Training Data

- Typical training data in machine translation

    - different types of corpora
        * European Parliament Proceedings
        * collection of movie subtitles
    - temporal structure in each corpus
    - similar sentences next too each other (e.g., same story / debate)

Shuffling the Training Data

- Typical training data in machine translation

  - different types of corpora
    - ∗ European Parliament Proceedings
    - ∗ collection of movie subtitles
  - temporal structure in each corpus
  - similar sentences next too each other (e.g., same story / debate)

- Online updating: last examples matter more

Shuffling the Training Data

- Typical training data in machine translation

  - different types of corpora
    * European Parliament Proceedings
    * collection of movie subtitles
  - temporal structure in each corpus
  - similar sentences next too each other (e.g., same story / debate)

- Online updating: last examples matter more

- Convergence criterion: no improvement recently

  $\rightarrow$ stretch of hard examples following easy examples: prematurely stopped

Shuffling the Training Data

- Typical training data in machine translation

  - different types of corpora
    * European Parliament Proceedings
    * collection of movie subtitles
  - temporal structure in each corpus
  - similar sentences next too each other (e.g., same story / debate)

- Online updating: last examples matter more

- Convergence criterion: no improvement recently

  $\rightarrow$ stretch of hard examples following easy examples: prematurely stopped

$\Rightarrow$ randomly shuffle the training data

(maybe each epoch)

Weight Initialization

- Initialize weights to random values

- Values are chosen from a uniform distribution

- Ideal weights lead to node values in transition area for activation function

Weight Initialization

- Initialize weights to random values

- Values are chosen from a uniform distribution

- Ideal weights lead to node values in transition area for activation function

- Input values in range $[-1; 1]$

$\Rightarrow$ Output values in range $[0.269; 0.731]$

For Example: Sigmoid

- Input values in range $[-1; 1]$

$\Rightarrow$ Output values in range $[0.269; 0.731]$

- Magic formula (n size of the previous layer)

$$\left[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}} \right]$$

- Input values in range $[-1; 1]$

$\Rightarrow$ Output values in range $[0.269; 0.731]$

- Magic formula (n size of the previous layer)

$$\left[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}} \right]$$

- Magic formula for hidden layers

$$\left[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}} \right]$$

  – n_j is the size of the previous layer
  – n_{j+1} size of next layer

Problem: Overconfident Models

- Predictions of the neural machine translation models are surprisingly confident

- Often almost all the probability mass is assigned to a single word
  (word prediction probabilities of over 99%)

Problem: Overconfident Models

- Predictions of the neural machine translation models are surprisingly confident

- Often almost all the probability mass is assigned to a single word
  (word prediction probabilities of over 99%)

- Problem for decoding and training

  - decoding: sensible alternatives get low scores, bad for beam search
  - training: overfitting is more likely

- Predictions of the neural machine translation models are surprisingly confident

- Often almost all the probability mass is assigned to a single word
  (word prediction probabilities of over 99%)

- Problem for decoding and training

  – decoding: sensible alternatives get low scores, bad for beam search
  – training: overfitting is more likely

- Solution: label smoothing

- Predictions of the neural machine translation models are surprisingly confident

- Often almost all the probability mass is assigned to a single word
  (word prediction probabilities of over 99%)

- Problem for decoding and training

  – decoding: sensible alternatives get low scores, bad for beam search
  – training: overfitting is more likely

- Solution: label smoothing

- Jargon notice

  – in classification tasks, we predict a *label*
  – jargon term for any output
  $\rightarrow$ here, we smooth the word predictions

Label Smoothing during Decoding

- Common strategy to combat peaked distributions: smooth them

- Recall

  - prediction layer produces numbers for each word
  - converted into probabilities using the softmax

$$p(y_i) = \frac{\exp s_i}{\sum_j \exp s_j}$$

Label Smoothing during Decoding

- Common strategy to combat peaked distributions: smooth them

- Recall

  - prediction layer produces numbers for each word
  - converted into probabilities using the softmax

$$p(y_i) = \frac{\exp s_i}{\sum_j \exp s_j}$$

- Softmax calculation can be smoothed with so-called **temperature** T

$$p(y_i) = \frac{\exp s_i/T}{\sum_j \exp s_j/T}$$

- Higher temperature $\rightarrow$ distribution smoother
  (i.e., less probability is given to most likely choice)

Label Smoothing during Training

- Root of problem: training

- Training object: assign all probability mass to single correct word

- Root of problem: training

- Training object: assign all probability mass to single correct word

- Label smoothing

  – truth gives some probability mass to other words (say, 10% of it)
  – uniformly distributed over all words
  – relative to unigram word probabilities
    (relative counts of each word in the target side of the training data)

adjusting the learning rate

- Gradient descent training: weight update follows the gradient downhill

- Actual gradients have fairly large values, scale with a learning rate
  (low number, e.g., $\mu = 0.001$)

- Change the learning rate over time

  – starting with larger updates
  – refining weights with smaller updates
  – adjust for other reasons

- Learning rate schedule

Momentum Term

- Consider case where weight value far from optimum

- Most training examples push the weight value in the same direction

- Small updates take long to accumulate

- Consider case where weight value far from optimum

- Most training examples push the weight value in the same direction

- Small updates take long to accumulate

- Solution: momentum term m_t

  – accumulate weight updates at each time step t
  – some decay rate for sum (e.g., 0.9)
  – combine momentum term m_{t-1} with weight update value Δw_t

$$m_t = 0.9 m_{t-1} + \Delta w_t$$

$$w_t = w_{t-1} - \mu \, m_t$$

- Common strategy: reduce the learning rate μ over time

- Initially parameters are far away from optimum $\rightarrow$ change a lot

- Later nuanced refinements needed $\rightarrow$ change little

- Now: different learning rate for each parameter

Adagrad

- Different parameters at different stages of training
  $\rightarrow$ different learning rate for each parameter

- Adagrad

  - record gradients for each parameter
  - accumulate their square values over time
  - use this sum to reduce learning rate

Adagrad

- Different parameters at different stages of training
  $\rightarrow$ different learning rate for each parameter

- Adagrad

  - record gradients for each parameter
  - accumulate their square values over time
  - use this sum to reduce learning rate

- Update formula

  - gradient $g_t = \frac{dE_t}{dw}$ of error E with respect to weight w
  - divide the learning rate μ by accumulated sum

$$\Delta w_t = \frac{\mu}{\sqrt{\sum_{\tau=1}^{t} g_\tau^2}} \, g_t$$

- Big changes in the parameter value (corresponding to big gradients g_t)
  $\rightarrow$ reduction of the learning rate of the weight parameter

Adam: Elements

- Combine idea of momentum term and reduce parameter update by accumulated change

- Momentum term idea (e.g., $\beta_1 = 0.9$)

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)g_t$$

- Accumulated gradients (decay with $\beta_2 = 0.999$)

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)g_t^2$$

- Initially, values for m_t and v_t are close to initial value of 0

- Adjustment

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \qquad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

- With $t \rightarrow \infty$ this correction goes away

$$\lim_{t \rightarrow \infty} \frac{1}{1 - \beta^t} \rightarrow 1$$

- Given

  - learning rate μ
  - momentum $\hat{m}_t$
  - accumulated change $\hat{v}_t$

- Weight update per Adam (e.g., $\epsilon = 10^{-8}$)

$$\Delta w_t = \frac{\mu}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

Batched Gradient Updates

- Accumulate all weight updates for all the training example → update (converges slowly)

Batched Gradient Updates

- Accumulate all weight updates for all the training example $\rightarrow$ update (converges slowly)

- Process each training example $\rightarrow$ update (stochastic gradient descent) (quicker convergence, but last training disproportionately higher impact)

Batched Gradient Updates

- Accumulate all weight updates for all the training example $\rightarrow$ update (converges slowly)

- Process each training example $\rightarrow$ update (stochastic gradient descent) (quicker convergence, but last training disproportionately higher impact)

- Process data in batches

  - compute all their gradients for individual word predictions errors
  - use sum over each batch to update parameters
  $\rightarrow$ better parallelization on GPUs

Batched Gradient Updates

- Accumulate all weight updates for all the training example → update

  (converges slowly)

- Process each training example → update (stochastic gradient descent)

  (quicker convergence, but last training disproportionately higher impact)

- Process data in batches

  – compute all their gradients for individual word predictions errors
  – use sum over each batch to update parameters
  → better parallelization on GPUs

- Process data on multiple compute cores

  – batch processing may take different amount of time
  – asynchronous training: apply updates when they arrive
  – mismatch between original weights and updates may not matter much

avoiding local optima

- One of hardest problem for designing neural network architectures and optimization methods

- Ensure that model converges to at least to a set of parameter values that give results close to this optimum on unseen test data.

- There is no real solution to this problem.

- It requires experimentation and analysis that is more craft than science.

- Still, this section presents a number of methods that generally help avoiding getting stuck in local optima.

Overfitting and Underfitting

- Neural machine translation models

  - 100s of millions of parameters
  - 100s of millions of training examples (individual word predictions)

- No hard rules for relationship between these two numbers

- Neural machine translation models

    - 100s of millions of parameters
    - 100s of millions of training examples (individual word predictions)

- No hard rules for relationship between these two numbers

- Too many parameters and too few training examples $\rightarrow$ overfitting

- Too few parameters and many training examples $\rightarrow$ underfitting

Regularization

- Motivation: prefer as few parameters as possible

- Strategy: set un-needed paramters a value of 0

Regularization

- Motivation: prefer as few parameters as possible

- Strategy: set un-needed paramters a value of 0

- Method

  - adjust training objective
  - add cost for any non-zero parameter
  - typically done with L2 norm

Regularization

- Motivation: prefer as few parameters as possible

- Strategy: set un-needed paramters a value of 0

- Method

  - adjust training objective
  - add cost for any non-zero parameter
  - typically done with L2 norm

- Practical impact

  - derivative of L2 norm is value of parameter
  - if not signal from training: reduce value of parameter
  - alsp called weight decay

- Not common in deep learning, but other methods understood as regularization

Curriculum Learning

- Human learning

  – learn simple concepts first
  – learn more complex material later

Curriculum Learning

- Human learning

  - learn simple concepts first
  - learn more complex material later

- Early epochs: only easy training examples

  - only short sentences
  - create artificial data by extracting smaller segments
    (similar to phrase pair extraction in statistical machine translation)
  - Later epochs: all training data

- Not easy to callibrate

- Training may get stuck in local optima

  – some properties of task have been learned
  – discovery of other properties would take it too far out of its comfort zone.

- Machine translation example

  – model learned the language model aspects
  – but cannot figure out role of input sentence

- Drop out: for each batch, eliminate some nodes

Dropout
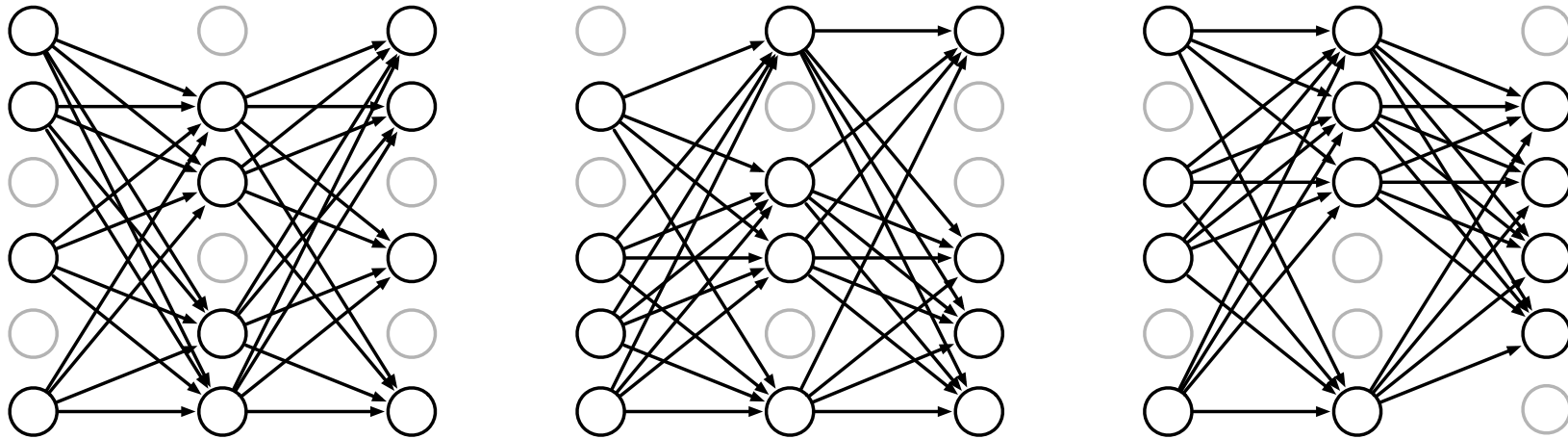
- Dropout

  - For each batch, different random set of nodes is removed
  - Their values are set to 0 and their weights are not updated
  - 10%, 20% or even 50% of all the nodes

- Why does this work?

  - robustness: redundant nodes play similar nodes
  - ensemble learning: different subnetworks are different models

Gradient Clipping

- Exploding gradients: gradients become too large during backward pass

$\Rightarrow$ Limit total value of gradients for a layer to threshold (τ)

- Use of L2 norm of gradient values g

$$L2(g) = \sqrt{\sum_j g_j^2}$$

- Adjust each gradient value g_i for each element i in the vector

$$g_i' = g_i \times \frac{\tau}{\max(\tau, L2(g))}$$

- During inference, average node values may become too large or too small

- Has also impact on training (gradients are multiplied with node values)

$\Rightarrow$ Normalize node values

- During training, learn bias layer

Layer Normalization: Math

- Feed-forward layer h^l, weights W, computed sum s^l, activation function

$$s^l = W \, h^{l-1}$$

$$h^l = \text{sigmoid}(h^l)$$

- Compute mean μ^l and variance σ^l of sum vector s^l

$$\mu^l = \frac{1}{H} \sum_{i-1}^{H} s_i^l$$

$$\sigma^l = \sqrt{\frac{1}{H} \sum_{i-1}^{H} (s_i^l - \mu^l)^2}$$

- Normalize s^l

$$\hat{s}^l = \frac{1}{\sigma^l}(s^l - \mu^l)$$

- Learnable bias vectors g and b

$$\hat{s}^l = \frac{g}{\sigma^l}(s^l - \mu^l) + b$$

- Deep learning: many layers of processing

⇒ Error propagation has to travel farther

- All parameters in processing change have to be adjusted

- Instead of always passing through all layers, add connections from first to last

- Jargon alert

  - shortcuts
  - residual connections
  - skip connections

- Feed-forward layer

$$y = f(x)$$

- Pass through input x

$$y = f(x) + x$$

- Note: gradient is

$$y' = f'(x) + 1$$

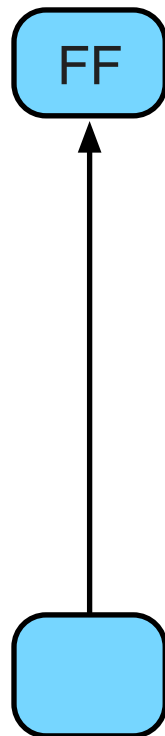- Constant $1 \rightarrow$ gradient is passed through unchanged

- Regulate how much information from $f(x)$ and x should impact the output y
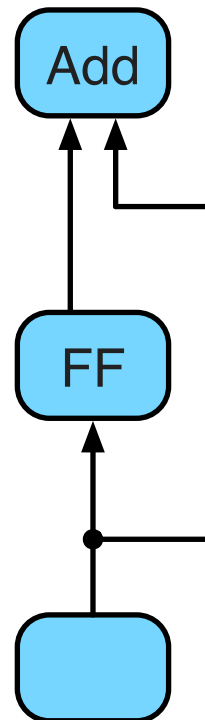
- Gate $t(x)$ (typically computed by a feed-forward layer)
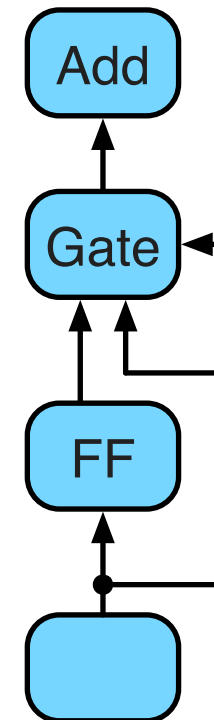
$$y = t(x) \ f(x) + (1 - t(x)) \ x$$

Basic Layer   Skip Connection   Highway Network

- Recall: Long short term memory (LSTM) cells

- Pass through of memory

$$\text{memory}^t = \text{gate}_{\text{input}} \times \text{input}^t + \text{gate}_{\text{forget}} \times \text{memory}^{t-1}$$

- Forget gate has values close to $1 \rightarrow$ gradient passed through nearly unchanged

generative adversarial training

- Traditional training

  - predict one word at a time
  - compare against correct word
  - proceed training with correct word

- Sequence-level training

  - predict entire sequence
  - measure translation with sentence-level metric (e.g., BLEU)

- May use n-best translations, beam search, etc.

- Game between two players

  - generator proposes a translation
  - discriminator distinguishes between generator's translation and human translation
  - generator tries to fool discriminator

- Training example: input sentence x and output sentence y

- Generator

  - traditional neural machine translation model
  - generates full sentence translations t for each input sentence

- Discriminator

  - is trained to classify (x, y) as correct example
  - is trained to classify (x, t) as generated example

1. First train generator to some maturity

2. Train discriminator on generator predictions and human reference translations

3. Train jointly

   – generator with additional objective to fool discriminator
   – discriminator to do well on detecting generator's output as such

- In practice, this is hard to callibrate correctly

Relationship to Reinforcement Learning

- No immediate feedback

  - chess playing: quality of move only revealed at end of game
  - walk through maze to avoid monsters and find gold

- Policy: decision process to which steps to take
  (here: generator, traditional neural machine translation model)

- Reward: end result
  (here: ability to fool discriminator)

- Popular technique: Monte Carlo search
  (here: Monte Carlo decoding)

- Training is called policy search