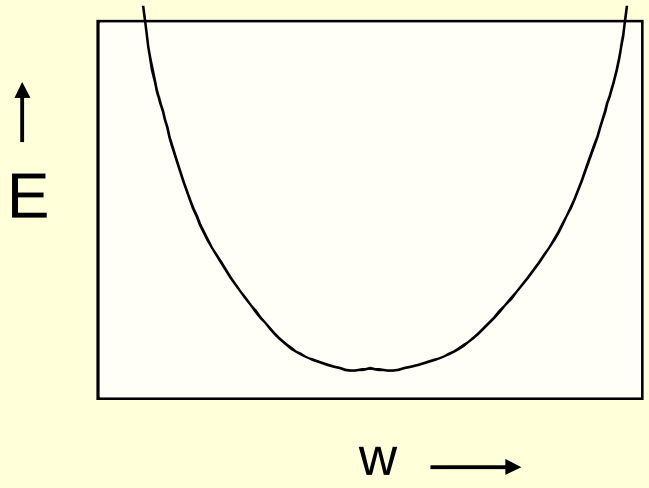
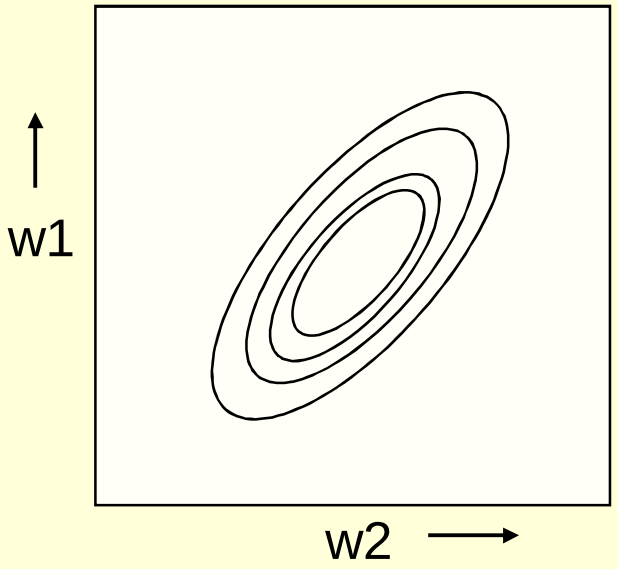
Neural Networks

Lecture 8
Modeling text using a recurrent neural network trained with a really fancy optimizer

The error surface for a linear neuron

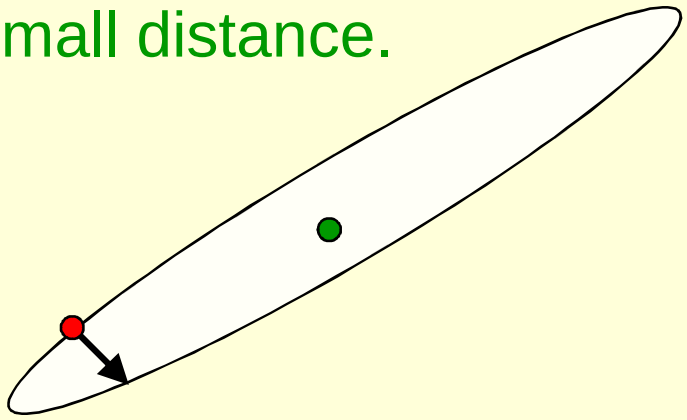
- The error surface lies in a space with a horizontal axis for each weight and one vertical axis for the error.
  - It is a quadratic bowl.
    - i.e. the height can be expressed as a function of the weights without using powers higher than 2. Quadratics have constant curvature (because the second derivative must be a constant)
  - Vertical cross-sections are parabolas.
  - Horizontal cross-sections are ellipses.





Convergence speed

- The direction of steepest descent does not point at the minimum unless the ellipse is a circle.
  - The gradient is big in the direction in which we only want to travel a small distance.



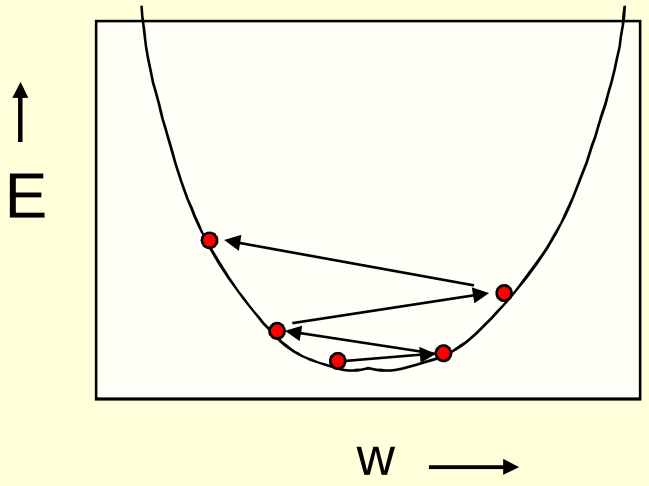
  - The gradient is small in the direction in which we want to travel a large distance.

$$\Delta w_i = -\varepsilon \frac{\partial E}{\partial w_i}$$

This equation is sick. The RHS needs to be multiplied by a term of dimension w^2 to make the dimensions balance.

How the learning goes wrong

- If the learning rate is big, it sloshes to and fro across the ravine. If the rate is too big, this oscillation diverges.
- How can we move quickly in directions with small gradients without getting divergent oscillations in directions with big gradients?

- 

Five ways to speed up learning

- Use an adaptive global learning rate
  - Increase the rate slowly if its not diverging
  - Decrease the rate quickly if it starts diverging
- Use separate adaptive learning rate on each connection
  - Adjust using consistency of gradient on that weight axis
- Use momentum
  - Instead of using the gradient to change the position of the weight "particle", use it to change the velocity.
- Use a stochastic estimate of the gradient from a few cases
  - This works very well on large, redundant datasets.
- Don't go in the direction of steepest descent.
  - The gradient does not point at the minimum.
    - Can we preprocess the data or do something to the gradient so that we move directly towards the minimum?

Newton's method

- The basic problem is that the gradient is not the direction we want to go in.
  - If the error surface had circular cross-sections, the gradient would be fine.
  - So maybe we should apply a linear transformation that turns ellipses into circles.
- Instead of turning the error surface into a circular bowl, we could achieve the same thing by transforming the gradient vector.
  - Transform it in exactly the way it would be transformed by turning the error surface into a circular bowl.

How to transform the gradient vector

- We would like to multiply the vector of gradients by the inverse of the curvature matrix.
  - This produces a vector that takes us straight to the minimum in one step for a quadratic surface.

$$\Delta w_i = -\frac{\partial E}{\partial w_i} H^{-1}$$

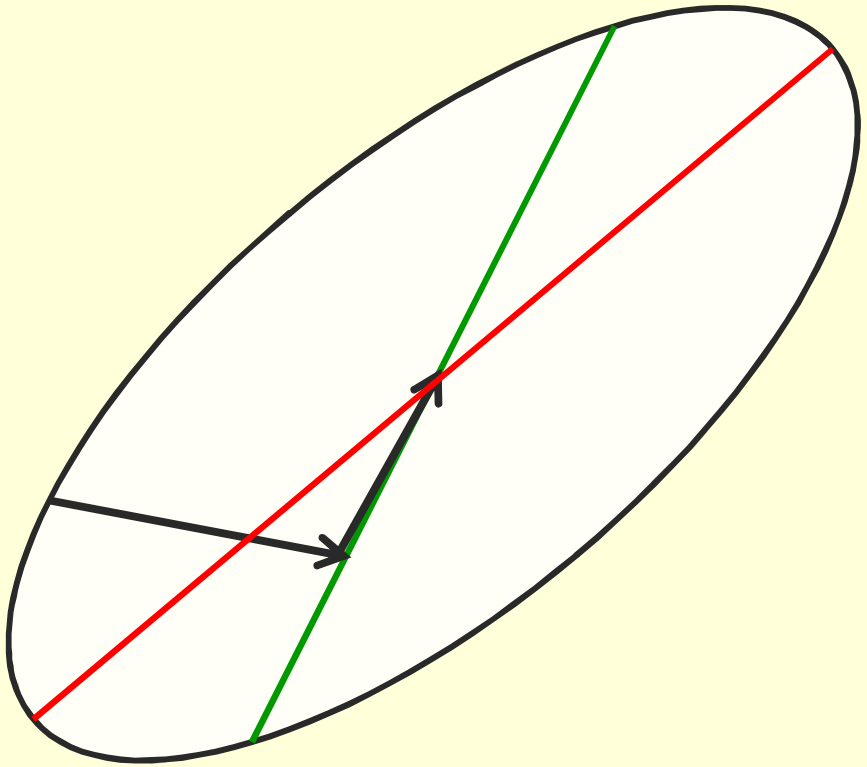
This equation is dimensionally correct.

- Unfortunately, the inverse curvature matrix has too many terms to be of use in a big neural network (the number of weights squared!)

Conjugate gradient

- There is an alternative to going to the minimum in one step by multiplying by the inverse of the curvature matrix.
- Use a sequence of steps each of which finds the minimum along one direction.
- Make sure that each new direction is “conjugate” to the previous directions.
  - This means that as you go in the new direction, you do not change the gradients in the previous directions.

A picture of conjugate gradient



The gradient in the direction of the first step is zero at all points on the green line.

So if we move along the green line we don’t mess up the minimization we already did in the first direction.

What does conjugate gradient achieve?

- After N steps, conjugate gradient is guaranteed to find the minimum of an N-dimensional quadratic surface.
  - After many less than N steps it has typically got the error to close to the minimum value.

- Conjugate gradient can be applied to a non-quadratic error surface and it usually works quite well, but the HF optimizer is much better.

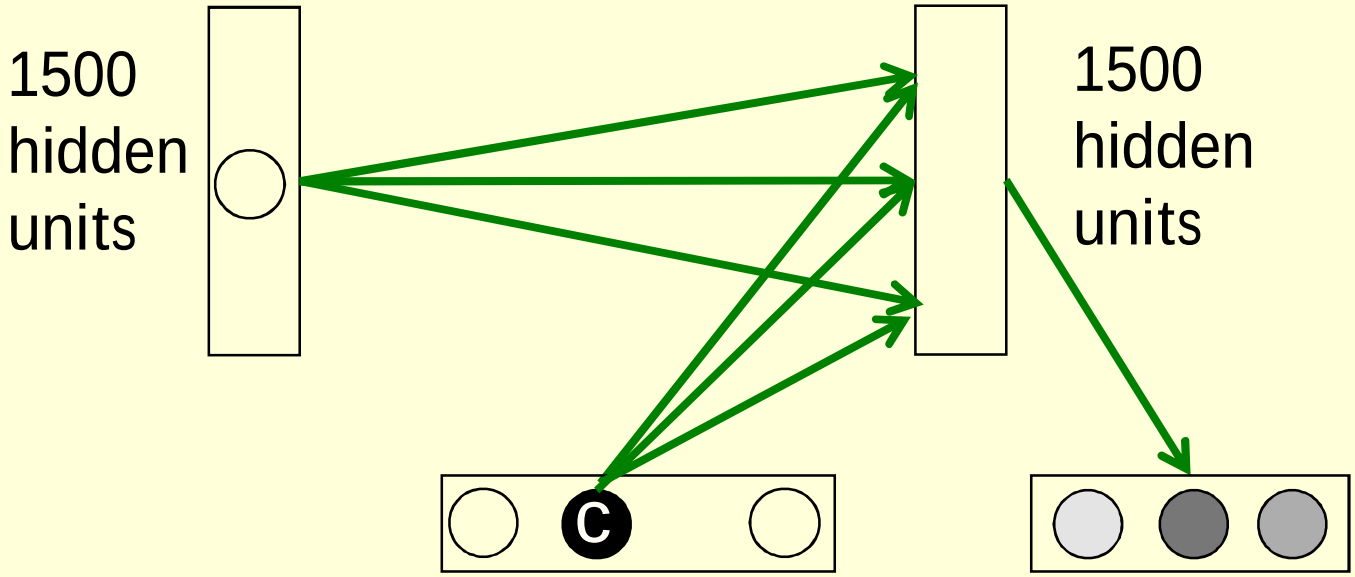
Training recurrent neural nets with a powerful optimizer

- Here is a good problem to demonstrate the power of RNNs: understand all the text on the web.
- The only way to predict the next word really well is to learn a model of what the source believes.
  - Stephen Harper suspended parliament because he wanted to avoid the danger of ….
- We are still a long way from learning this level of understanding!
  - But machine learning will get there eventually and programming rules by hand never will.

Advantages of working with characters

- The web is composed of character strings.
- Any learning method powerful enough to understand the world by reading the web ought to find it trivial to learn which strings make words.
- Pre-processing text to get words is a big hassle
  - What about prefixes, suffixes etc.
  - What about New York?
  - What about subtle effects like “sn” words?

The obvious recurrent neural net

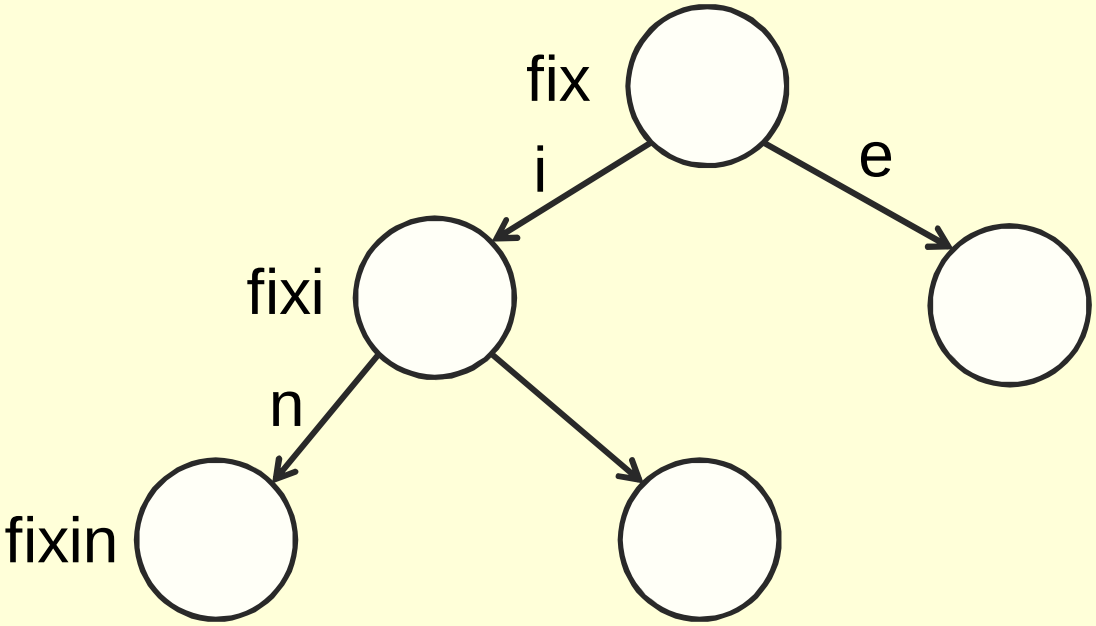


character:
1-of-86

predicted distribution
for next character.

It is a lot easier to
predict 86 characters
than 100,000 words.

Another view of the recurrent net



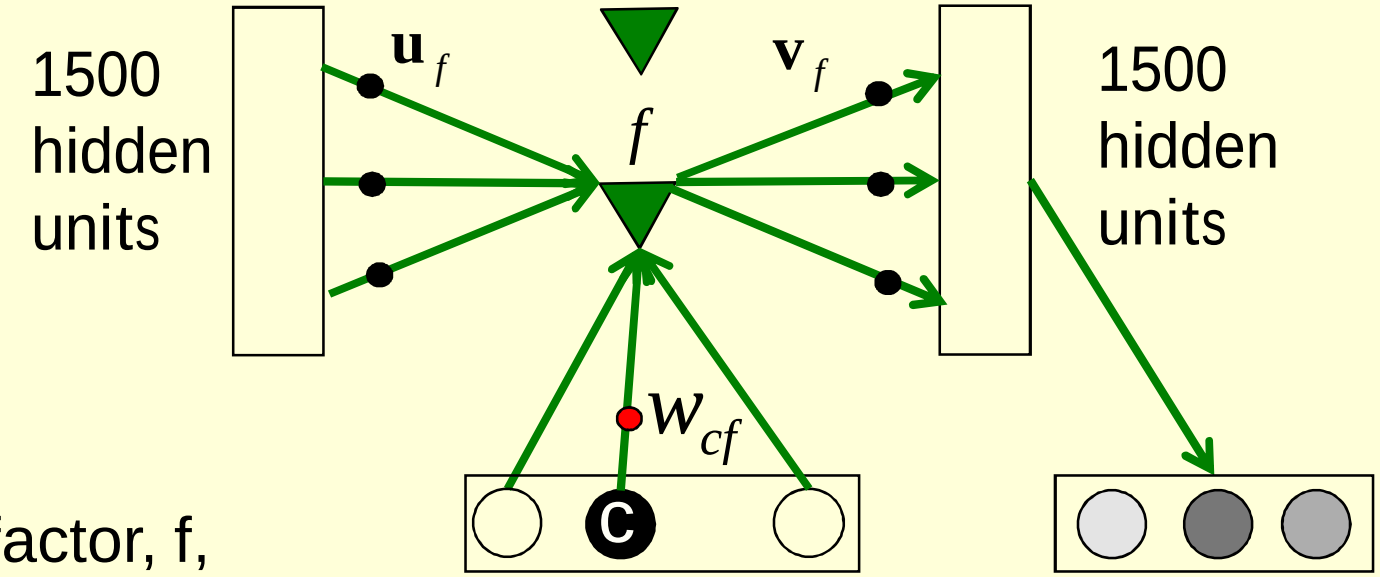
Each node is a hidden state vector. The next character must transform this to a new node.

- There are exponentially many nodes!
- Different nodes can share structure because they use distributed representations.
- The next hidden representation needs to depend on the conjunction of the current character and the current hidden representation.

Multiplicative connections

- Instead of using the inputs to the recurrent net to provide additive extra input to the hidden units, we could use the current input character to choose the whole hidden-to-hidden weight matrix.
  - But this requires 86x1500x1500 parameters
  - This would make the net overfit.
- Can we achieve the same kind of multiplicative interaction using fewer parameters?
  - Is there some way to allow the 86 character-specific weight matrices to share parameters?

Using a few thousand 3-way factors to allow a character to create a whole transition matrix



Each factor, f, defines a rank one matrix , $\mathbf{u}_f \mathbf{v}_f^T$

character: 1-of-86

predicted distribution for next character.

Each character, c, determines a gain w_{cf} for each of these rank one matrices

Training the character model

- Ilya Sutskever used about a million strings of 250 characters taken from wikipedia. For each string he starts predicting at the 51st character.
- It takes 5 days on 8 GPU boards each with 240 processors to get a really good model. It needs very big mini-batches (its lucky he didn't start in 1980)
- Ilya's best model beats the state of the art for character prediction, but works in a very different way from the best other models.
  - It can balance quotes and brackets over long distances. The rival models cannot do this.

Some text generated by the model

In 1974 Northern Denver had been overshadowed by CNL, and several Irish intelligence agencies in the Mediterranean region. However, on the Victoria, Kings Hebrew stated that Charles decided to escape during an alliance. The mansion house was completed in 1882, the second in its bridge are omitted, while closing is the proton reticulum composed below it aims, such that it is the blurring of appearing on any well-paid type of box printer.